UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E DE ESTATÍSTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM AMBIENTE OPERACIONAL PARA TEMPO-REAL BASEADO NO MODO VIRTUAL 8086 DOS PROCESSADORES 80386 E 80486

por

Sidney Flores

Dissertação submetida à Universidade Federal de Santa Catarina para a obtenção do grau de mestre em Ciência da Computação

Prof. Luiz Fernando Jacintho Maia, Dr.
Orientador

Florianópolis, Setembro de 1995.

Um Ambiente Operacional para Tempo-Real baseado no Modo Virtual 8086 dos processadores 80386 e 80486

Sidney Flores

Essa dissertação foi julgada adequada para obtenção do Título de

Mestre em Ciência da Computação

Especialidade Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação

Prof. Luiz Fernando Jacintho Maia, Dr.
Orientador

Prof. Rogério Cid Bastos, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Luiz Fernando Jacintho Maia, Dr.
Orientador (Presidente)

Prof. Vitório Bruno Mazzola, Dr.

Prof. José Wagner Maciel Kaehler, Dr.

Prof. Antonio Augusto R. Coelho, Dr.

A meus pais

Alvaro Flores (*in memorian*) e
Ulysses Shaurich Flores

Agradecimentos

Antes de qualquer outro agradecimento, devo expressar minha gratidão à minha esposa *Eunice* e à minha filha *Fabiana* pela compreensão, carinho e apoio durante essa longa jornada.

Ao *Prof. Dr. Hermann Adolf Harry Lücke*, que orientou esse trabalho durante sua concepção e desenvolvimento, suas fases mais importantes. Sua falta entre nós só é aplacada pela lembrança do carinho com que nos tratava, pela maneira respeitosa com que corrigia nossos erros e nos indicava os caminhos a seguir e pela solidez dos ensinamentos que nos transmitia em todos os instantes do curto espaço de tempo em que nossas vidas estiveram rumando juntas.

Ao *Prof. Dr. Luiz Fernando Jacintho Maia* que gentilmente acedeu assumir a orientação deste trabalho em sua fase final, pelo esforço e dedicação que nos deferiu.

Ao amigo *Isaac Benjamim Benchimol*,companheiro de pesquisa na fase embrionária deste trabalho, meu agradecimento pelas agradáveis e proveitosas discussões em torno da arquitetura dos processadores 80386/486.

Aos demais colegas de curso, pelo apoio e amizade durante nossa convivência. A falta do lar foi suportável apenas pelo convívio com eles.

À *Vera Lúcia*, que nos deu suporte nas aflições burocráticas e sociais durante todo o tempo em que estivemos ligados ao CPGCC, meus mais carinhosos agradecimentos.

À UFSC que me acolheu durante o periodo do curso e propiciou os meios necessários à execução deste trabalho. Agradeço também à *Companhia Estadual de Energia Elétrica* pela oportunidade de participar deste curso.

Aos colegas de trabalho, que tiveram sua carga aumentada durante a minha ausência, meus sinceros agradecimentos pelo inconveniente e minha promessa de compensá-los com maior dedicação ao trabalho e com mais amizade e companheirismo no dia-a-dia.

Agradecimento Especial

Este trabalho é dedicado ao Prof. Hermann Adolf Harry Lücke, que o orientou durante sua quase totalidade. À ele, que durante todo o tempo de desenvolvimento do curso, durante todo o tempo de concepção e projeto do presente trabalho e, mesmo durante o período da enfermidade que o tirou de nosso convívio, jamais deixou de conduzir e incentivar nossos passos, creditamos tudo o que de valor possa constar desta dissertação.

Sidney Flores

Resumo

Esse trabalho apresenta um panorama dos sistemas operacionais multitarefa para tempo real com aplicações em automação industrial, descreve as características de sistemas operacionais multitarefas, as necessidades das aplicações em tempo real, traça um panorama sobre os processadores 80386/486 e suas características para apoio à multitarefa e desenvolve um ambiente operacional multitarefa para tempo real cuja característica principal é ser capaz de executar tarefas desenvolvidas para o ambiente DOS.

Abstract

This report presents a survey of multitask real time operating systems with applications in industrial automation, describes the characteristics of multitask operating systems, the needs of real time applications, presents a general view of 80386/80486 processors and their characteristics that aids multitask and develops a real time multitask environment whose main characteristic is be able to execute tasks developed for DOS environment.

Indice Analítico

Índice de Figuras

Glossário

GDT General Descriptor Table - Tabela Geral de Descritores - Estrutura de dados usadas pelos processadores 80386/80486, operando no modo protegido, que contém descritores de segmentos. Junto com os descritores da LDT, definem todo o espaço de endereçamento válido para os programas em execução.

LDT Local Descriptor Table - Tabela Local de Descritores - Estrutura de dados usadas pelos processadores 80386/80486, operando no modo protegido, que contém descritores de segmentos. Junto com os descritores da GDT, definem todo o espaço de endereçamento válido para os programas em execução.

GDTR Global Descriptor Table Register - Registrador da Tabela Global de Descritores - Registrador dos processadores 80386/80486 que apontam a posição da GDT.

LDTR LocalDescriptor Table Register - Registrador da Tabela Local de Descritores - Registrador dos processadores 80386/80486 que apontam a posição da LDT.

TSS Task State Segment - Segmento de Estado da Tarefa - Segmento de memória com atributos especiais e utilização no gerenciamento de tarefas. É uma das estruturas de dados de apoio à multitarefa fornecido pela arquitetura dos processadores 80386/80486

PD Page Directory - Diretório de Páginas - Estrutura de dados utilizada pelo mecanismo de paginação. É o primeiro nível de tradução dos endereços lineares em endereços físicos.

PT Page Table - Tabela de Páginas - Estrutura de dados utilizada pelo mecanismo de paginação. É o segundo nível de tradução dos endereços lineares em endereços físicos.

PDBR Page Directory Base Register - Registrador de Base do Diretório de Páginas - Registrador que aponta para a base do diretório de páginas ativo.

PTE Page Table Entry - Entrada de Tabela de Páginas - É cada uma das estruturas de dados que constitui a Tabela de Páginas. As informações indicam a posição do início da página na memória física e as informações de proteção e gerência de paginação que se aplicam à página apontada por esta PDE.

PDE Page Directory Entry - Entrada de Diretório de Páginas - É cada uma das estruturas de dados que constitui o Diretório de Páginas. As informações indicam a posição do início da tabela de página na memória física e as informações de proteção e gerência de paginação que se aplicam a todas as páginas apontadas pelas PTEs constantes desta PDE.

DPL Descriptor Privilege Level - Nível de Privilégio do Descritor - Indica o nível de privilégio do segmento descrito por este descritor de segmento.

CPL Current Privilege Level - Nível de Privilégio do Corrente - Indica o nível de privilégio do segmentode código atualmente sendo executado.

RPL Requested Privilege Level - Nível de Privilégio Necessário - Indica o nível de privilégio do segmento necessário para acesso a um segmento de código.

IDT Interrupt Descriptor Table - Tabela de Descritores de Interrupção - Estrutura de dados do modo protegido, semelhante ao vetor de interrupções do DOS, que descreve cada rotina de atendimento às interrupções no modo protegido.

IDTR Interrupt Descriptor Table Register - Registrador da Tabela de Descritores de Interrupção - Regeistrador que indica a posição na memória da Tabela de Descritores de Interrupção.

PIC Programable Inteerrupt Controller - Controlador de Interrupção Programável - Componente da placa-mãe dos computadores que assiste às interrupções de hardware através da geração do sinal de interrupção no processador e indicação do número da interrupção ocorrida.

Kb Kilobytes - Unidade correspondente a 1024 bytes.

Mb Megabytes - Unidade correspondente a 1048576 bytes. (1048576 = 1024 Kb)

Gb Gigabytes - Unidade correspondente a 1073741824 bytes. (1073741824 = 1024 Mb)

XDOS DOS Extendido - Ambiente Operacional que permite ao DOS gerenciar processos concorrentes, fornecendo serviços adequados às aplicações de tempo real.

PCTR Protocolo de Comunicação em Tempo Real - Conjunto de funções que permitem comunicação em nível de rede entre computadores usando XDOS.

SDD Sistema de Distribuição de Dados - Sistema executando com XDOS e PCTR de maneira da concentrar e distribuir os dados coletados no campo para as aplicações que precisarem.

V86 Virtual 86 - Modo de operacão dos procesadores 80386 e 80486, mais adequadamente, submodo do modo protegido, que simula um processador 8086, mas mantém características de proteção e paginação.

1. Introdução

1.1 Panorama Atual

A tendência atual em sistemas de software para automação industrial aponta para arquiteturas organizadas por camadas. Nestas camadas, encontramos o núcleo multitarefa do sistema operacional e, para apoio às atividades de interface, o uso intensivo dos serviços de sistemas operacionais conhecidos, utilizados como base das implementações.

A organização em camadas facilita o desenvolvimento porque incentiva a adoção de características de modularidade para o sistema operacional. A modularidade, por sua vez, permite a definição clara da interface entre os módulos ou as camadas. Esta definição precisa das interfaces entre camadas simplifica correções, adaptações e ampliação das capacidades do sistema. Resumindo, a arquitetura por camadas é uma técnica atual de engenharia de software aplicada aos sistemas operacionais.

O uso de um sistema operacional conhecido (DOS, Windows, OS/2 e outros tantos) para base de um sistema multitarefa facilita o desenvolvimento da interface através da qual o sistema disponibilizará seus serviços para as aplicações, porque permite o uso de técnicas de programação já conhecidas e o uso de compiladores comerciais, também familiares, para desenvolvê-las.

O número de camadas e a distribuição de suas funções depende da implementação específica, sendo típica uma organização como a descrita na Fig. 1, onde destacamos a camada de disposição, camada de coordenação e camada de execução [7].

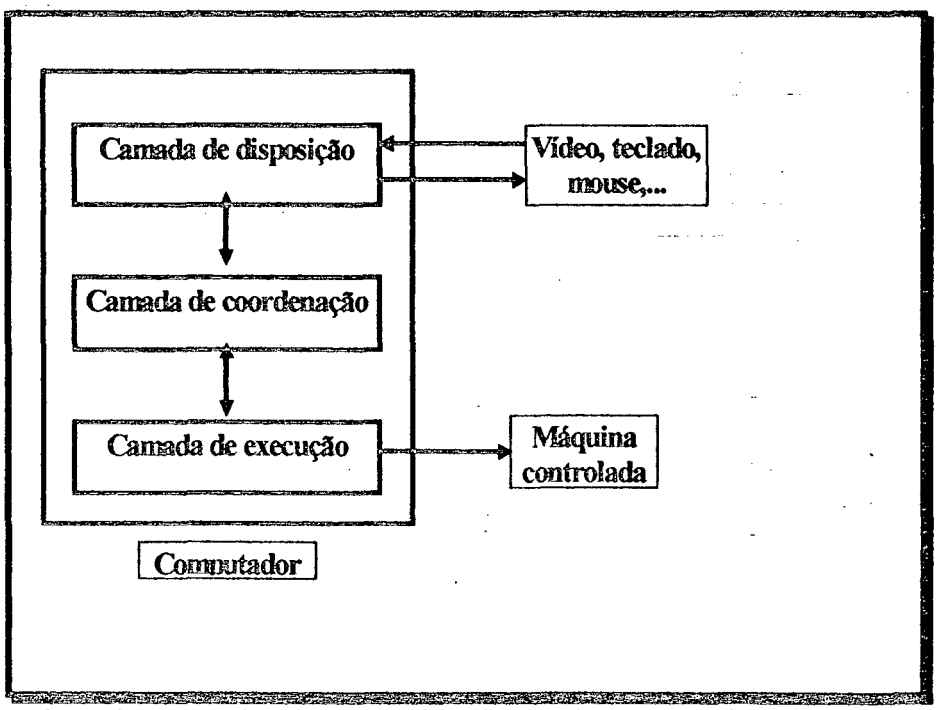

Fig. 1.1 - Organização típica, em camadas, de um sistema multitarefa para tempo real.

A camada de disposição é onde residem os programas aplicativos, como:

- Interface homem-máquina, que auxilia a operação das máquinas do sistema que está sendo controlado.
- Programas que implementam interfaces com periféricos e com redes de computadores.

Nesta camada é usado freqüentemente o sistema operacional DOS ou o Windows com extensões que objetivam suprir deficiências inerentes de seu projeto original, adaptando-os para uso multitarefa. O largo uso destes sistemas é devido ao conhecimento sobre os mesmos que os usuários acumularam ao longo de anos de uso, e pelo fato de serem comuns no parque instalado de máquinas tipo PC. No caso particular do Windows, o grande uso se deve à estética da interface gráfica, além da facilidade de uso devida à padronização desta interface, comum a todos os aplicativos. Essa camada, portanto, é a que auxilia o operador a dispor do computador, das máquinas controladas e dos equipamentos a elas interligados, de onde o vem nome de camada de disposição.

Na camada de coordenação, encontramos o núcleo multitarefa para tempo real, que se constitue no coração do sistema. Esse núcleo deve dar ao sistema operacional nativo as facilidades que ele não tem por questões de projeto inicial. Tais facilidades podem ser:

- Capacidade de gerenciar várias tarefas de forma preemptiva.
- Capacidade de usar toda a memória disponível no equipamento.
- Capacidade de permitir que as tarefas troquem informações entre si.
- Capacidade de permitir a sincronização entre processos.

Essa camada é a que coordena a utilização dos recursos do equipamento em que os processos estão executando concorrentemente e por isso é chamada de coordenação.

Na camada de execução temos a interface entre o equipamento computacional onde os processos são executados e os equipamentos que controlam o sistema físico. Aqui temos todos os programas específicos para interfaceamento e uso de sensores, atuadores e acesso a redes industriais de aquisição de dados e de controle.

Um exemplo mais concreto de um sistema operacional multitarefa em tempo real baseado em 80386/486 usando na camada de disposição DOS ou Windows pode ser visto em [31] e será descrito sucintamente a seguir.

No exemplo, a memória contém, em sua parte baixa, o DOS, área para programas DOS e área para memória de vídeo e EPROMS, chegando até o final do primeiro megabyte. Após este limite, foi reservada uma área de um megabyte para memória estendida e expandida, neste caso, para uso do Windows. Acima, no topo da memória rodam o núcleo multitarefa do sistema operacional e as tarefas concorrentes. Na Fig. 1.2, pode ser visto, no lado esquerdo, a distribuição de memória.

O esquema funcional do exemplo, que pode ser visto no lado direito da Fig. 1.2, deixa aparecer claramente a camada de disposição onde vemos uma representação do núcleo multitarefa para modo protegido que se sobrepõe aos sistemas operacionais DOS e Windows e aos programas dependentes do 'hardware', incluindo o BIOS da máquina. Programas DOS e Windows escritos para serem executados concorrentemente, acessam os serviços deste núcleo através de sua interface (por exemplo, através de mensagens ou de interrupções de software) ou acessam diretamente serviços do DOS, do Windows ou do BIOS, através das interrupções de software que estes sistemas normalmente usam para oferecer seus serviços.

A camada de coordenação contém o núcleo multitarefa no modo protegido. Essa camada não tem acesso externo, pois sua função é tão somente gerenciar o uso dos recursos do equipamento e o controle de despacho de tarefas. É aqui que se situa o escalonador, os gerenciadores de processo, os gerenciadores de memória e os gerenciadores de outros recursos que o sistema disponibilize como mensagens, semáforos, temporizadores, entre outros.

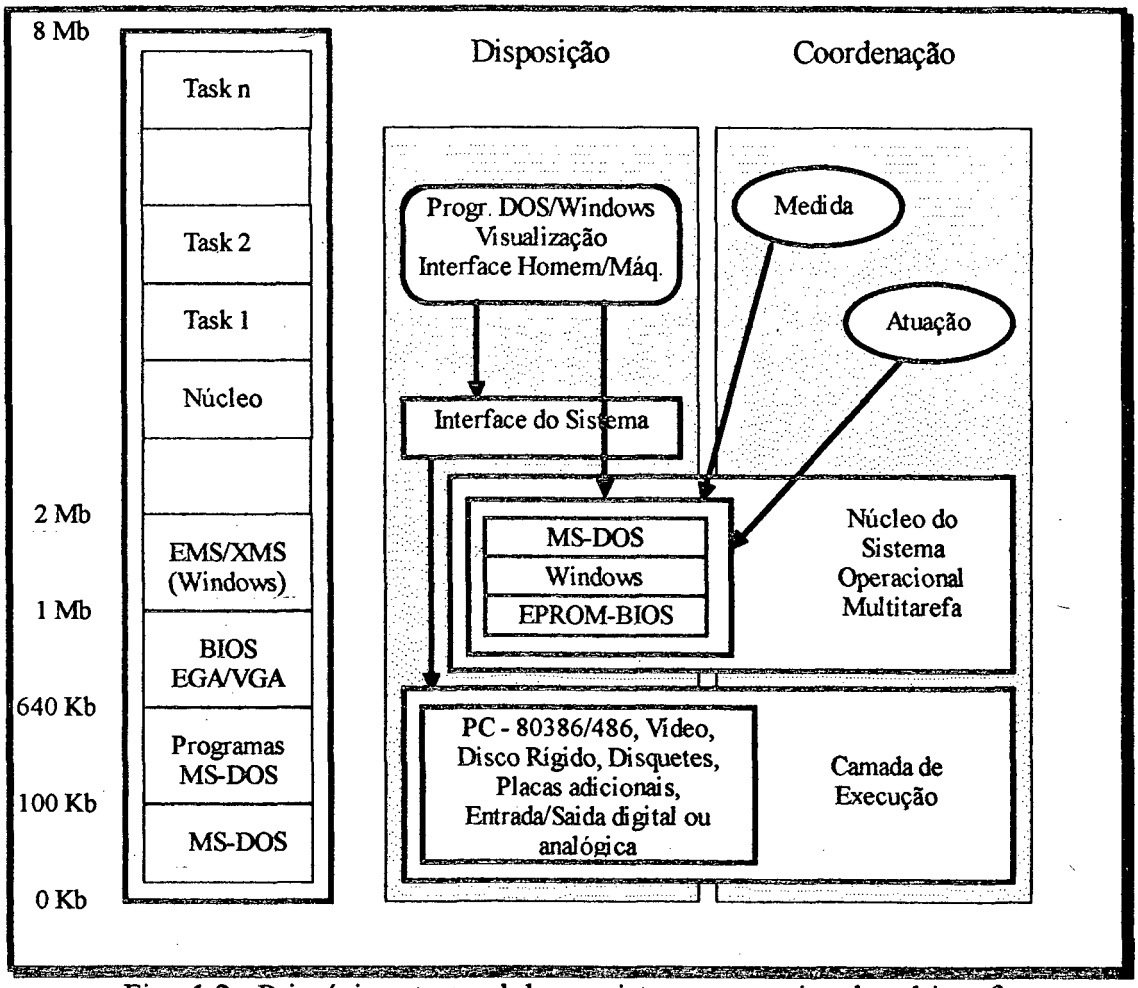

Fig. 1.2 - Princípio estrutural de um sistema operacional multitarefa

A camada de execução é usada para permitir que os programas DOS e Windows, que foram desenvolvidos para executar concorrentemente, possam acessar os recursos de hardware através dos serviços do núcleo multitarefa disponíveis na interface com a camada de disposição. Um exemplo seria o acesso a facilidades de rede através dos serviços de troca de mensagens.

Do exposto verificamos que existe uma tendência atual de criar sistemas operacionais multitarefa para tempo real, com aplicações em automação industrial, contemplando os seguintes aspectos principais:

- Uso de hardware de uso comum e de baixo custo, como computadores padrão IBM-PC e seus compatíveis.
- Uso de sistemas operacionais bem conhecidos, simples e de baixo custo sobre os quais se desenvolvem aplicativos concorrentes para interfacear com os usuários e com os recursos computacionais.

- Acréscimo de núcleos multitarefas a estes sistemas operacionais para dotá-los das características necessárias à multitarefa em tempo real.
- Arquitetura do sistema operacional em camadas, com interfaces e funções bem definidas, criando modularidade o que facilita ampliações do sistema de acordo com as necessidades e facilidade de depuração.

Cabe ainda esclarecer que o acréscimo de núcleos multitarefa para tempo real é necessário, no caso do DOS, por ele ser monotarefa e não ter capacidade de gerenciar tarefas concorrentes, além de não ser determinístico, ou seja, não se pode determinar o tempo em que uma função será executada, pela forma como trata as interrupções. No caso do Windows, sistema que é multitarefa, a necessidade do núcleo é devida ao fato de sua multitarefa não ser preemptiva e também por não ser determinístico.

Os sistemas de tempo real devem permitir que se determine com precisão os tempos de execução de funções. Isso é difícil porque não se pode determinar o instante em que uma interrupção será acionada durante a execução da função, porque as interrupções são eventos assíncronos. Além disto, o tempo entre a ocorrência da interrupção e o início do seu atendimento pela função adequada do sistema operacional (tempo de latência no atendimento a uma interrupção) é difícil de ser determinado com precisão porque o sistema pode estar atendendo a uma outra interrupção de maior prioridade. O último intervalo de tempo envolvido, o tempo de execução da função do sistema operacional que atende à interrupção, também é quase impossível de ser determinado porque, durante o mesmo, podem ocorrer outras interrupções de prioridade maior, suspendendo temporariamente a execução da atual.

Com relação aos sistemas operacionais mencionados até agora como sistema operacional nativo na camada de disposição, citamos principalmente o DOS e o Windows. Isso se deve à estreita ligação destes sistemas com os processadores 80x86 da Intel que são os processadores alvos de nossa implementação.

Não podemos deixar de mencionar a incapacidade do DOS de acessar toda a memória instalada nos computadores atuais equipados com os processadores 80386/80486. Historicamente, o DOS foi projetado para acesso a 1 Mb de memória e disponibilidade para os programas aplicativos, de um trecho que inicia acima da faixa usada para os dados do DOS e código residente associado ao sistema e termina em 640 Kb.

Essa implementação quebra essa restrição através do uso de máquinas virtuais que compartilham os trechos de sistema e que permitem que se coloque código executável em qualquer posição da memória. Isso é possível através do uso do modo protegido dos processadores 80386/80486, ou mais particularmente, do modo virtual 8086. Esse modo alia as facilidades de paginação do modo protegido com a facilidade de programação do modo real, modo em que o DOS executa, sendo muito adequado para nossas intenções.

De tudo o que foi exposto, concluimos que nossa implementação se justifica:

- Pelo amplo uso dos processadores 80386/80486 em aplicações de automação industrial.
- Pelo uso de um sistema conhecido, DOS, de fácil utilização, como base dos desenvolvimentos.
- Pela eliminação da principal restrição do DOS, que é a impossibilidade do uso efetivo da totalidade da memória instalada no computador.

No que se refere a prazos, consideramos nosso desenvolvimento como situado no médio prazo devido à disponibilidade atual destes processadores e do sistema operacional (DOS) que servirá de base para o sistema multitarefa.

No longo prazo, esboçam-se desenvolvimentos com base em outros processadores como, por exemplo, o Alpha da Digital Equipment Co. ou da série 600 de processadores da associação entre a IBM e a Motorola [12]. Os sistemas operacionais previstos para estes processadores baseiam-se em núcleos 'micro-kernel', incluídos no próprio projeto do processador, que permitem a implementação fácil de várias interfaces para sistemas operacionais como OSF/1, VMS, WindowsNT, OS/2 e UNIX. Um exemplo para este tipo de 'micro-kernel' é o sistema MACH [26].

1.2 Motivação e Objetivos do Trabalho

1.2.1 Histórico

Um exemplo da aplicação das tendências que descrevemos no item anterior, está na Companhia Estadual de Energia Elétrica (CEEE) do Rio Grande do Sul [23] [32].

Desde 1976, a CEEE opera seu sistema elétrico com o apoio de um Sistema de Supervisão e Controle (SSC) fabricado pela Leeds&Northrup. Esse sistema de supervisão era constituído, inicialmente, por uma estação central e por 16 estações terminais remotas. A estação central é composta por um sistema dual de minicomputadores modelo MAC16 da Lockeed, ligados a um sistema de interface homem-máquina e aquisição de dados modelo C2050 da Leeds&Northrup. As estações remotas inicialmente instaladas são, também, de fabricação da Leeds&Northrup modelo C2020, tendo atualmente evoluído para 20 unidades com adição de estações remotas fabricadas pela Elebra.

Com o aumento de complexidade do sistema elétrico a ser supervisionado, decorrente da sua expansão, o Sistema de Supervisão e Controle teve sua capacidade esgotada tanto no aspecto de instalação de novas estações terminais remotas como no acréscimo de novas funções em software que se tornavam cada vez mais necessárias à modernização da operação do sistema elétrico. Essa impossibilidade de acréscimo se dá por falta de recursos de memória e pela incapacidade de processamento em tempo viável. Ao mesmo tempo, a manutenção deste equipamento passou a ser cada vez mais difícil devido à obsolescência dos materiais e à dificuldade de se conseguir os componentes necessários, muitos dos quais já fora de fabricação.

Assim, tornou-se necessária a substituição deste Sistema de Supervisão e Controle por um outro de maior capacidade e de tecnologia mais moderna. Para tanto, foram elaboradas as especificações funcional e técnica de um novo sistema e foi lançada uma licitação nacional para sua aquisição. Paralelamente foram feitas adaptações ao Sistema de Supervisão e Controle para suprir necessidades emergenciais, como adição de duas unidades de disco rígido removível de 5 Mb cada uma, comunicação serial com terminais remotos de supervisão regional desenvolvidos com microcomputadores Apple, interligação, à nível nacional, ao Sistema Interligado de Supervisão Automática da Secretaria do Grupo Coordenador para a Operação Interligada, no Rio de Janeiro e, mais recentemente, a interligação com o Centro Nacional de Operação do Sistema (CNOS), em Brasília.

Esse contrato de fornecimento não foi efetivado devido ao custo de cerca de 15 milhões de dólares e às condições adversas de se obter o financiamento do projeto, na época.

A partir daí, decidiu-se pelo desenvolvimento de um novo Sistema de Supervisão e Controle e o primeiro passo foi a implantação da Rede de Microcomputadores da Operação e o desenvolvimento de vários programas com o objetivo de cobrir as funções de pré- e pós-operação que não eram atendidas pelo sistema anterior. O segundo passo foi modernizar o sistema de supervisão regional substituindo os terminais remotos de supervisão, de tecnologia Apple, por micros PC386 dotados de monitores VGA.

Com o objetivo de criar os meios necessários a esse desenvolvimento, a CEEE enviou, à Florianópolis, o Eng. Sandro Rocha Peres para fazer o Curso de Pós-graduação em Computação na Universidade Federal de Santa Catarina. O objeto de seu trabalho de conclusão de curso foi, então, o desenvolvimento do Sistema de Distribuição de Dados [23] e sua interligação ao Sistema de Supervisão e Controle. Esse projeto incluiu o desenvolvimento de um ambiente operacional multitarefa, denominado XDOS, voltado a comunicação de dados em tempo real, um protocolo de comunicação para tempo real, denominado PCTR, e um aplicativo gerenciador da base de dados e distribuidor de dados.

O XDOS, um núcleo multitarefa adicionado ao sistema operacional MS-DOS, é um exemplo de como se pode usar um sistema operacional conhecido e dotá-lo de características adicionais para permitir seu uso como multitarefa em tempo real. Seu projeto também obedece à tendência de arquitetura por camadas, conforme veremos adiante.

1.2.2 Implementação Atual e Previsão de Adições Futuras

A Rede de Supervisão Regional foi, então implantada utilizando microcomputadores PC com processadores 80386, substituindo os equipamentos Apple. Todos os enlaces eram feitos via serial utilizando as interfaces assíncronas originais dos microcomputadores com exceção dos minicomputadores MAC-16 do Sistema de Supervisão e Controle antigo, para os quais foi necessário desenvolver uma placa multi-função contendo 2 interfaces seriais assincronas de alta velocidade, uma expansão de memória de 256 Kb e um microprocessador Z80 para gerenciar as funções da mesma.

Sobre o software de suporte (DOS, XDOS, PCTR) foram desenvolvidos, além do Sistema de Distribuição de Dados (SDD), dois outros módulos:

- O módulo chamado de Cabeça de Console armazena os dados recebidos do SDD, executa processamento de alarmes e processamento dos pontos calculados.
- O módulo de interface homem-máquina permite que o operador interaja com o sistema através de mouse, teclado e vídeo.

Esta é a situação atual do sistema, estando em desenvolvimento software e algum hardware, não disponível comercialmente, para fazer comunicação com unidades terminais remotas.

O objetivo futuro da CEEE é chegar a uma configuração em que a rede de tempo real, rede interna do sistema de supervisão e controle, possua os seguintes elementos:

- Computadores para comunicação com unidades remotas, que farão contato com 32 estações remotas para coleta de dados nas principais subestações do sistema elétrico. Para tanto já foi desenvolvido hardware e firmware baseado no microprocessador 8031 para adaptar o protocolo de comunicação das estações terminais remotas, de diversos fabricantes como Leeds&Northrup e Elebra, à conexão serial dos PCs, via modems. O

software necessário à essa função também já foi desenvolvido e está em período de testes.

- Computadores para distribuição de dados, elementos que já estão em operação, sendo um deles configurado como principal e outro como reserva.
- Computadores para processamentos de informações de entradas/saidas e tempo/freqüência. Por entrada/saida entendemos o equipamento que acionará os elementos do painel mímico e fará supervisão de grandezas e estados do próprio equipamento de supervisão e do ambiente do Centro de Operação do Sistema. Por tempo/freqüência estamos nos referindo a um cartão desenvolvido pela CEEE e baseado no processador 8031, já em operação, que, instalado no interior de um PC, faz medição, digitalização e registro de perturbações da freqüência do sistema.
- Computadores cabeça de console e controle automático da geração que, conforme já descrito encontra-se parcialmente desenvolvido e em operação, faltando apenas a implementação do processo que executará o algoritmo de controle automático da geração.
- Computador para estimador de estados cujo algoritmo executa configuração da rede, modelagem da rede configurada e estimação de estados de grandezas não supervisionadas. Esse elemento já se encontra em operação em tempo real.
- Computadores para interface homem-máquina que já se encontram em operação.
- Computadores para uso como console remota que já se encontram em operação nos Centros Regionais de Operação.
- Computador para comunicação com o CNOS da Eletrobrás em Brasília, que se encontra hoje em desenvolvimento.

Por sua vez, os serviços de apoio à operação do sistema terão à sua disposição uma rede chamada 'on-line' que já está ligada à rede de tempo real através de um computador, chamado de Servidor de Tempo Real, que disponibiliza o acesso aos dados obtidos da rede de tempo real para estes serviços e, no sentido inverso, o fornecimento de dados de estudos de pré- e pós-operação para os algoritmos que rodam nas máquinas da rede de tempo real.

1.2.3 Motivação

Porém, o aumento da complexidade e da quantidade dos módulos que devem executar concorrentemente em uma máquina, no esquema de multitarefa, esgotou o espaço de endereçamento disponível ao DOS. Isso obrigou a usar mais computadores ligados em rede para tarefas que poderiam ser executadas com apenas um computador, caso não estivéssemos obrigados a nos restringir à faixa de endereçamento do DOS. Se fosse possível aumentar o espaço de endereçamento para acomodar mais código e mais dados, um computador poderia acomodar mais processos e eliminar tráfego na rede e dispêndio extra com equipamento. Aumentar a capacidade para dados é possível com o auxílio dos emuladores de memória expandida existentes no mercado, porém a quantidade de dados manipulada, à excessão das estruturas de dados pertinentes ao programa gerenciador do banco de dados, não é grande o bastante para liberar quantidades de memória significativas.

A solução que se vislumbra é aumentar a capacidade de endereçamento do sistema operacional para executar código na memória acima de um megabyte. Para tanto, devemos usar o modo virtual 8086 dos processadores 80386 para desenvolver um ambiente

operacional aos moldes do XDOS e compatível com o mesmo para suplantar essa limitação de memória, podendo, através da paginação, utilizar toda a memória presente na máquina.

1.3 O XDOS

O XDOS iniciou a ser desenvolvido em 1988 pelo Eng. Sandro Rocha Peres, como parte de seu trabalho de conclusão do Curso de Pós-Graduação em Computação na Universidade Federal de Santa Catarina [23], o XDOS é um software residente que acrescenta primitivas de concorrência, sincronização, temporização e comunicação para uso no desenvolvimento de aplicativos em tempo real ao sistema operacional DOS. Suas características principais são:

- Arquitetura em camadas. A camada de coordenação é constituida pelo núcleo que gerencia a troca de tarefas. A camada de disposição é constituída pelo próprio DOS e pelos programas aplicativos. A camada de execução é constituída por rotinas de acesso aos recursos do equipamento, como as rotinas para comunicação serial, rotinas para uso dos recursos de rede Ethernet, rotinas para acesso à unidades terminais remotas via canais de comunicação e outros já existentes no próprio DOS, como acesso a arquivos.
- Interface da camada de coordenação com a camada de disposição definida por chamadas de sistema através de interrupção de "software" (INT 60H).
- Processos definidos como rotinas desenvolvidas sob DOS em uma linguagem de alto nível e ativados pelo programa principal, utilizando os serviços do núcleo mutitarefa do sistema.
- Gerenciamento da execução concorrente de até 62 processos de usuário, utilizando prioridades estáticas.
- Implementa semáforos contadores, com capacidade de contagem de até 127 operações sucessivas, para sincronização e controle de acesso a regiões críticas.
- Permite troca de mensagens, com tamanhos variáveis e limitados a até 64 Kbytes entre processos, mesmo que eles se encontrem em máquinas diferentes da rede.
- Controle dos recursos de comunicação disponíveis no equipamento, implementando para cada um deles um protocolo de enlace.
- Implementa recursos de temporização, permitindo acionamento de processos temporizados, "time-outs" e temporizações internas nos protocolos de comunicação.
- O intervalo de tempo utilizado para executar cada processo é de 55 mseg, unidade fornecida pelo temporizador da máquina. O tempo de chaveamento resultou, em testes, ser de 2 mseg, dando uma performance de 95% de tempo usado em processamento útil.
- Processos classificados nos estados PRONTO, quando apto a executar, BLOQUEADO, quando à espera de algum recurso, mensagem ou semáforo e EXCLUÍDO, quando executou a primitiva de término e não deverá voltar a rodar a menos que seja novamente ativado por outro processo.
- Algoritmo de escalonamento por prioridades estáticas com uma fila para cada prioridade. Dentro de cada fila de prioridade, o algoritmo de escalonamento é o da fila circular ("round robin").

- Não existe proteção entre os processos concorrentes, pois o DOS não permite nenhum esquema de proteção.

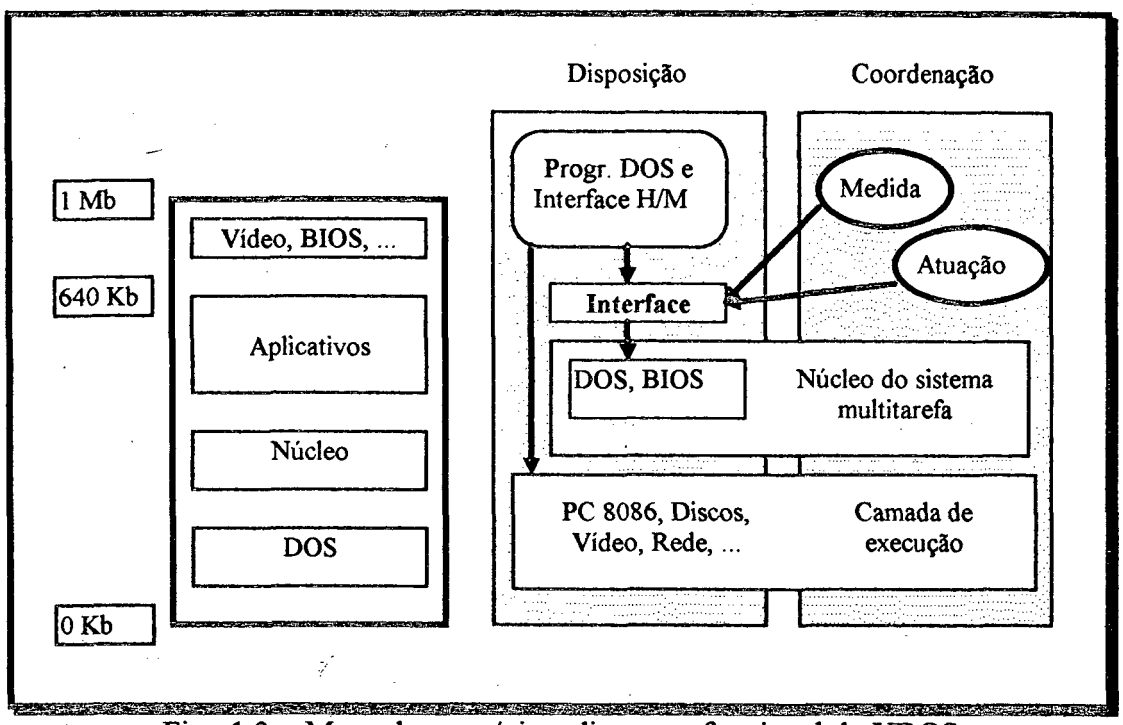

Fig. 1.3 - Mapa de memória e diagrama funcional do XDOS

Conforme se pode verificar por esta descrição sumária, o XDOS é uma extensão do sistema operacional nativo DOS ao qual acrescenta funções para execução concorrente de processos. Possui as mesmas limitações do DOS, entre as quais destacamos, por ser o objeto deste trabalho, a impossibilidade de utilizar a memória disponível nas máquinas acima de 1 Mb.

1.4 - Conteúdo

Este trabalho desenvolve um ambiente multitarefa para tempo real compatível com o XDOS a nível de aplicativo utilizando o modo virtual 86 dos processadores 80386/80486. O objetivo de tal desenvolvimento é ultrapassar a maior limitação do XDOS, herdada do DOS, que é a restrição de endereçamento. Utilizamos o modo protegido para abrigar o núcleo de um sistema multitarefa que gerencia a execução de várias tarefas no modo virtual 8086. O uso do modo virtual 8086 se prende à facilidade de desenvolvimento de programas porque utiliza as técnicas já conhecidas de desenvolvimento de programas para DOS. Para poder usar a memória disponível na máquina foi usada a capacidade de paginação dos processadores 80386/80486. Assim é possível colocar as tarefas virtuais 8086 em qualquer ponto da memória apesar de parecer, a cada uma delas, que está executando no primeiro megabyte do espaço de endereçamento físico e sob o DOS.

A compatibilidade foi mantida para que as aplicações para supervisão em tempo real que já foram desenvolvidas e que já estão executando com sucesso no Centro de Operação do Sistema da CEEE, continuem a executar, sem necessidade de modificações, no novo sistema.

As vantagens deste desenvolvimento são:

- Aumento de performance, em virtude do uso de endereçamento de 32 bits no núcleo do sistema, agilizando os acessos à memória.

- Aumento da segurança dos processos devido ao uso de proteção, no sentido de que cada um terá sua área de processamento e cada área é inacessível aos outros processos. Isso impedirá que um processo venha a prejudicar outro por estouro de pilha ou ponteiros errados.
- Economia causada pela necessidade de menos máquinas para o sistema completo. Com o uso do XDOS, ao preencher o primeiro megabyte de memória com processos, se torna necessário agregar mais uma máquina à rede para incluir nela os outros processos necessários. A previsão de necessidade de serviços futuros, conforme visto no final do ítem anterior, mostra que ainda existem muitos serviços a serem implantados. O aumento do número de processos que se pode executar em uma mesma máquina reduz as necessidades futuras de equipamento.
- Conhecimento dos métodos de programação para o modo protegido e modo virtual 8086 dos processadores 80386/80486, particularmente no que se refere ao uso das características de apoio à multitarefa e paginação.

As experiências com sistemas de 'micro-kernel', apesar de mais gerais, ainda não estarão disponíveis por algum tempo, de forma que o desenvolvimento do sistema é válido atualmente. Este sistema será de utilidade para qualquer entidade ou indivíduo com necessidade de usar um sistema multitarefa para tempo real nos processadores 80386/80486 utilizando o DOS.

2. Ambiente Operacional Multitarefa para Tempo Real

Esse trabalho descreve a implementação de um ambiente operacional multitarefa para tempo real usando os processadores 80386/486 em seu modo virtual 8086 de operação. Assim, discutiremos inicialmente os conceitos principais de sistemas operacionais multitarefa para tempo real.

De acordo com suas características, os sistemas operacionais podem ser classificados em:

- **Multiusuário** - quando o sistema operacional possui a capacidade de atender a mais de um usuário simultaneamente.
- **Multitarefa** - são aqueles sistemas operacionais que tem a capacidade de executar mais de um programa ou mais de um trecho, de um mesmo programa, de forma concorrente.
- **Multiprocessamento** - são os sistemas operacionais capazes de atender ao gerenciamento de mais de um processador em um mesmo computador, executando um ou mais programas ao mesmo tempo.

No caso de um sistema operacional multitarefa para tempo real, as qualidades que se espera dele, além de gerenciar recursos, controlar processamento simultâneo de várias tarefas e se comunicar com dispositivos externos é de que seja capaz de responder rapidamente a eventos que ocorram aleatoriamente no tempo. Além disso, este tempo deve ser plenamente definido, o que se costuma chamar de tempos determinísticos. Cada tarefa, cuja função pode ser atender a um evento particular, ou pode ser prover uma capacidade funcional ou pode ser, ainda, resolver um problema específico, executa de maneira completamente independente das outras tarefas, mas estão todas relacionadas pelas primitivas do sistema operacional.

Outra característica importante do sistema é a forma como o escalonador procede para parar uma tarefa que está executando e indicar qual a tarefa que será executada a seguir. Existem vários algoritmos que são usados para tratar desta escolha e do mecanismo de retirar o processador de um processo e dá-lo a um outro e os mais importantes serão objeto de discussão adiante. O que nos importa mais neste instante é reforçar a necessidade e a existência de uma parte do sistema operacional multitarefa que se encarrega de tomar conta destas funções..

2.1 Processo

Um conceito-chave para sistemas operacionais é o conceito de processo ou tarefa [2][27]. Um processo é basicamente um programa sendo executado. Ele consiste de um código executável, de dados, da pilha, do indicador de instrução, do indicador de pilha, dos valores contidos nos demais registradores do processador e todas as demais informações necessárias para executar o programa. Cada processo se comporta como se tivesse sua própria CPU, embora, na verdade, uma única CPU troque de um processo para outro, a intervalos regulares de tempo.

2.2 Áreas Críticas

É comum em sistemas multitarefa que dois processos, que estejam executando independentemente, necessitem acessar uma área comum, uma variável compartilhada ou ainda

um arquivo que ambos utilizem em suas tarefas. Neste caso, pode acontecer um problema típico destes sistemas. Pode ser que uma tarefa esteja dentro desta área comum, processando suas informações, quando o sistema operacional a suspende, passando o controle para a próxima tarefa a ser executada. A próxima tarefa, então, acessa também a área comum e a encontra em um estado não coerente, ou seja, como o primeiro processo a acessá-la não executou todo seu processamento, pode não ter ajustado todos os ponteiros, índices ou outros dados necessários para a seqüência normal do processamento.

Uma maneira clássica de resolver esse impasse é garantir exclusão mútua no acesso à área crítica. A exclusão mútua significa que, enquanto um processo estiver acessando a área crítica, esteja ele ativo ou bloqueado, nenhum outro processo poderá acessá-la. Existem várias formas de se garantir a exclusão mútua: desabilitar interrupções para impedir que o sistema operacional tome controle; testar uma variável que indique se alguém está executando dentro da região crítica; controlar que cada um entre só quando for sua vez e outras mais. Porém, todas estas tem a indesejável característica de ocupar o processador enquanto estão esperando.

Para evitar esta característica foram desenvolvidos outros métodos. O primeiro deles é bloquear a execução de um processo enquanto espera que outro, que já está na região crítica, termine seu processamento e o reative. Outro processo são os semáforos que são contadores do número de desbloqueamentos executados em uma região crítica controlada pelo semáforo. Os semáforos contadores evitam os desencontros no número de bloqueamentos e desbloqueamentos que acontecem no caso de semáforos comuns e que geram processos que jamais são desbloqueados.

2.3 Escalonamento

Quando um ou mais processos estão em condições de serem executados, o sistema operacional deve decidir qual deles o será. A parte do sistema operacional que toma essa decisão é chamada de escalonador e o algoritmo que o escalonador emprega para determinar o processo que será executado a seguir é chamado de algoritmo de escalonamento. O algoritmo de escalonamento pode ter vários níveis de detalhes desde um muito simples até outros bastante complexos.

As principais propriedades dos algoritmos de escalonamento são:

- **Justiça** - essa propriedade busca tornar garantido que cada processo tenha o mesmo direito de acesso à CPU.
- **Eficiência** - cujo objetivo é procurar manter a CPU ocupada pelos processos todo o tempo, de maneira que, sempre que haja processos a executar, a CPU não esteja ociosa.
- **Tempo de resposta** - no caso de haver usuários interativos no sistema, buscamos, por essa propriedade, minimizar o tempo de resposta sentido por estes usuários.

Um dos primeiros algoritmos de escalonamento, ainda da época dos processamentos em lote era executar o próximo programa da pilha de cartões e este programa era executado até ter completado seu processamento ou ser cancelado por alguma violação de segurança. Tal tipo de escalonamento é conhecida como não-preemptiva.

Algoritmos onde o sistema operacional suspende a execução de um processo para passar a executar outro que esteja apto a ser executado chama-se escalonamento preemptiva. Entre os métodos de escalonamento preemptiva, existem vários algoritmos de escalonamento

possíveis, dentre os quais destacamos pela simplicidade, facilidade de implementação e largo uso, os seguintes:

- **Escalonamento por fila circular** ('round robin')
- **Escalonamento por prioridade**

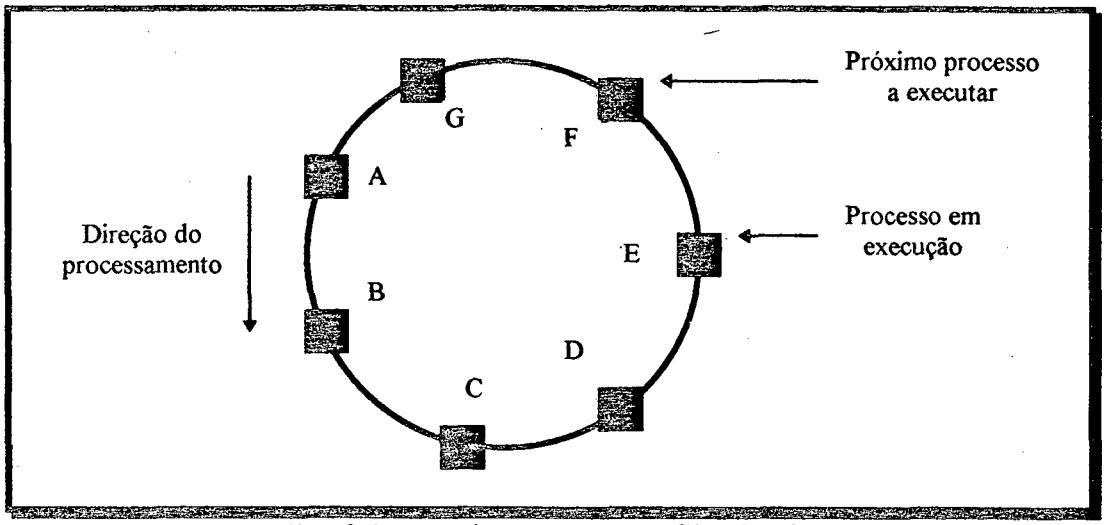

Fig. 2.1 - Escalonamento por fila circular

Um dos mais antigos, simples, confiáveis e mais usados algoritmos é o da fila circular. Neste algoritmo, que está ilustrado na Fig. 2.1, cada processo tem um intervalo de tempo em que lhe é permitido executar. Se o processo ainda estiver rodando no final de seu intervalo de tempo, a CPU lhe é retirada e dada a outro processo. Se o processo bloqueou ou terminou antes de ter atingido o fim de seu intervalo de tempo, a CPU também é dada ao próximo processo na fila. Nesse processo existe embutida a suposição de que todos os processos possuem a mesma prioridade.

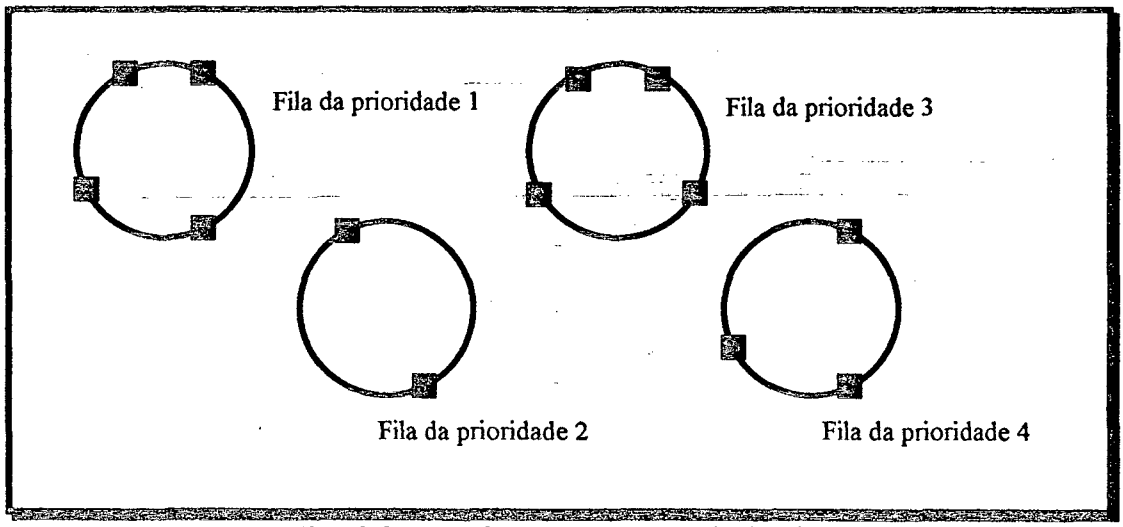

Fig. 2.2 - Escalonamento por prioridades

No algoritmo de escalonamento por fila circular, a escolha do próximo processo a executar é simplesmente tomar o próximo processo da fila dos processos que estão aptos a

executar. A escolha do intervalo de tempo a ser dado a cada processo para executar deve ser tal, que o tempo usado para trocar de processo não seja comparável, em ordem de grandeza, com o intervalo de tempo para os processos executarem. A eficiência é tida como o tempo útil de execução de processos, frente ao tempo total de utilização da CPU.

Um outro processo bastante usado é a escalonamento por prioridade, que se pode ver na Fig. 2.2, onde existe uma fila de processos para cada nível de prioridade do sistema e, dentro de cada fila, a escolha é baseada na fila circular. A escolha de um processo em uma fila de menor prioridade só é possível quando não existirem mais processos aptos a executar nas filas de prioridades mais elevadas.

2.4 Sincronização

Num sistema multitarefa em tempo real é necessário, por vezes, sincronizar processos. Casos típicos são etapas de fabricação em paralelo onde cada fase não leva exatamente o mesmo tempo. No final dos dois ramos, quem terminou antes sua parte tem que aguardar que o outro termine a sua para que um terceiro, então, assuma a seqüência da fabricação. Os processos mais comuns de sincronização, e os mais empregados, são os semáforos e a troca de mensagens. Semáforos também são empregados para garantir exclusão mútua às áreas críticas.

2.4.1 Semáforos

Um semáforo [27] é uma variável protegida que só pode ser acessada através de duas operações chamadas P e V, além de uma operação de inicialização.

A operação P testa um semáforo e verifica se seu valor é positivo. Se for positivo, a operação P decrementa o semáforo e continua a sequência normal de processamento. Se o valor for zero ou negativo, o processo é bloqueado. O teste do valor, alterá-lo e, possivelmente bloquear o processo é feito em uma operação única, indivisível, de maneira a garantir que uma vez que a operação sobre um semáforo comece, nada possa interromper essa operação até que ela seja concluída.

A operação V incrementa o valor do semáforo e, se um ou mais processos estiverem bloqueados nele, incapazes de prosseguir seu processamento, um deles é desbloqueado e prosseguirá seu processamento quando for escalado pelo sistema operacional. A escolha do processo a ser desbloqueado é feita normalmente pela ordem na fila associada ao semáforo, ou seja, o primeiro a ser bloqueado será o primeiro a ser desbloqueado. A operação V também é executada de maneira indivisível, desde o incremento do semáforo até o desbloqueamento do processo. Nenhum processo bloqueia ao executar uma operação V e pode bloquear ao executar uma operação P.

Além de garantir a exclusão mútua a um trecho crítico de programa, os semáforos são de grande valia na sincronização de processos e no gerenciamento de recursos escassos do sistema. No caso da sincronização, quando o primeiro alcança o semáforo, fica bloqueado nele até o outro processo chegar no semáforo e desbloqueá-lo, ficando assim os dois processos sincronizados.

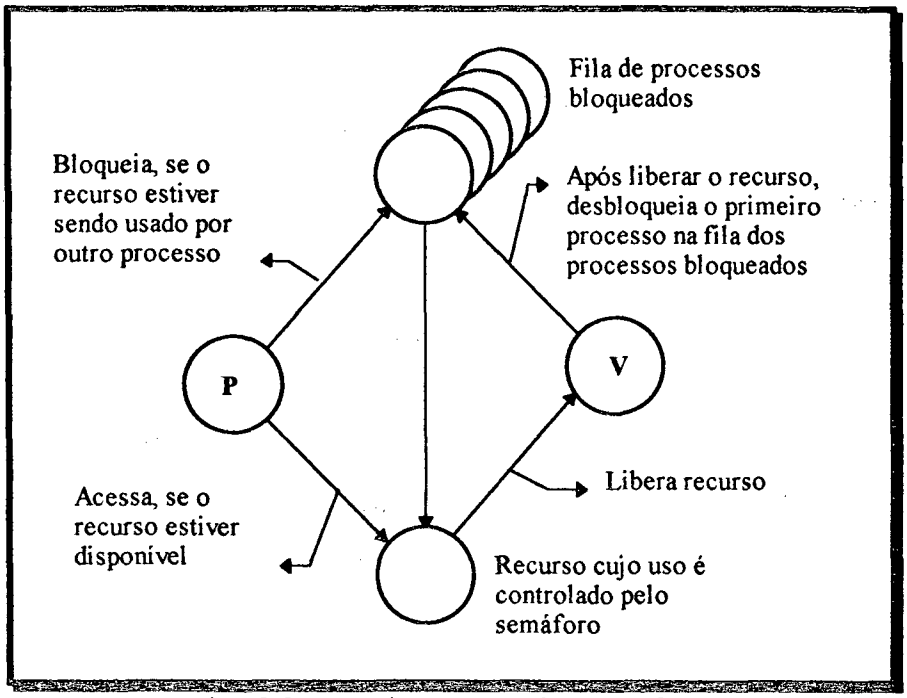

Fig. 2.3 - Funcionamento dos semáforos

2.4.2 - Troca de Mensagens

É fundamental em um sistema operacional multitarefa para tempo real que dois processos possam trocar mensagens entre si. A troca de mensagens entre processos, além de servir para partilhar dados, serve para sincronização de processos. Durante a sincronização, o primeiro processo a atingir certo ponto do seu processamento, bloqueia à espera de uma mensagem que só chegará quando o outro processo a ser sincronizado atingir um outro ponto em seu próprio processamento em que enviará a mensagem esperada pelo primeiro, sincronizando-os.

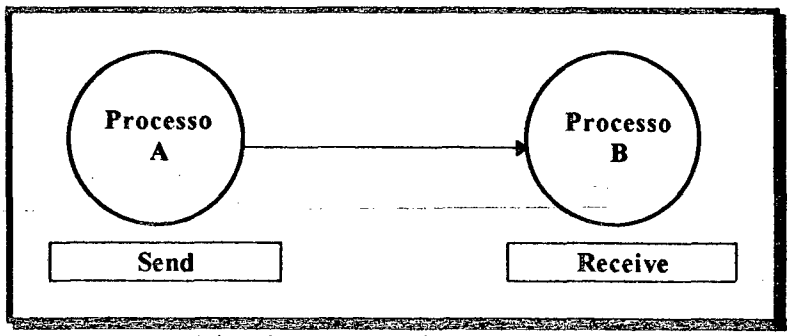

Fig. 2.4 - Troca de mensagens

Existem vários algoritmos para a comunicação entre processos, como 'rendez-vous' e caixas postais ('mailboxes'). No primeiro algoritmo, um dos processos que quer se comunicar bloqueia até que o outro processo também esteja em condições de conectar para completar a comunicação. Assim, os dois precisam estar em pontos determinados de seus processamentos para que a comunicação se efetive. No segundo algoritmo, o processo que vai enviar a mensagem, coloca-a em uma caixa postal e segue adiante em seu processamento. Quando o processo que vai receber a mensagem a quiser, poderá retirá-la da caixa postal, não havendo, portanto, sincronização.

A troca de mensagens é necessária não apenas a processos que estejam executando na mesma máquina. Processos executando em máquinas diferentes devem poder se comunicar através de uma rede, se a aplicação exigir que mais de um computador seja utilizado para executá-la. Neste caso, pode-se incorrer em problemas como perda de mensagens. Para garantir que as mensagens serão recebidas, pode-se lançar mão de confirmações de recebimento que, por sua vez, também podem ser perdidas. Enfim, temos que tomar cuidados especiais com todos os problemas de transmissão de dados através de rede.

2.5 Temporização

Em sistemas operacionais multitarefas de tempo real, o uso de temporizadores é de grande valia no caso de implementação de tempos de espera ('time-outs') ou na ativação de rotinas periódicas. O controle da temporização é conseguido através de relógios, que são elementos criados pelo sistema operacional na forma de estruturas de dados que associam um contador de tempo a um processo. O processo temporizado fica bloqueado até que o tempo, definido na inicialização do relógio, chegue ao fim. Nesse ponto o processo é desbloqueado e volta a esperar sua vez de executar. Se o processo for de alta prioridade, executará imediatamente após ser desbloqueado, se não for, o tempo ficará indefinido.

2.6 - Tempo Real

Um sistema operacional multitarefa para tempo real é aquele é capaz de responder a eventos específicos em um tempo definido. Os eventos são normalmente interrupções de hardware. Os sistemas operacionais multitarefa projetados para executar aplicações em tempo real necessitam, então, de características especiais no que tange ao tempos de execução dos processos e ao atendimento de interrupções.

As interrupções tem necessidade de serem atendidas em um intervalo de tempo definido no projeto do sistema. O tempo entre a ocorrência da interrupção e o início da resposta do computador a esse evento é chamado de tempo de latência da interrupção. O tempo de latência da interrupção é uma medida para avaliação de computadores para uso em aplicações de tempo real. Segundo esta medida, o DOS é excelente para tempo real, pois o hardware é padrão e a grande quantidade de documentação sobre como programá-lo permite que funções sejam ligadas diretamente a interrupções de hardware diminuindo grandemente o tempo de atendimento.

O tempo de atendimento do evento, entre o início e o final do processo que atende o evento deve ser definido e ocorrer dentro de um intervalo de tempo especificado de acordo com o projeto do sistema, baseado nas necessidades da aplicação.

Os tempos de execução de um processo, independentemente das interrupções que possam ocorrer durante sua execução, tem que ser determinado na sua periodicidade e em sua duração. Isso quer dizer que, para cada processo, deve ter perfeitamente definido o maior intervalo de tempo em que ele será executado. Além disso deve ser possível determinar a maior duração da execução do processo.

Todas essas restrições são difíceis de se obter ao mesmo tempo, porque possuem características antagônicas, ou seja, quando se busca eliminar uma delas, se prejudica

diretamente as demais. Portanto, é necessário que se tente obter um sistema que seja o melhor possível, para determinada aplicação, ajustando as características de acordo com o caso.

3. Modos de Operação dos Processadores 80386 e 80486

Tendo visto algumas características principais dos sistemas operacionais multitarefa para tempo real, passamos agora a discutir um pouco as características dos processadores 80386/486, relevantes para este sistema. Esses processadores se constituem no hardware que pretendemos empregar para desenvolvimento de nosso sistema.

Os processadores 80386/80486 são capazes de operar em três modos distintos. O primeiro deles é o modo real, modo em que iniciam a operação ao serem ligados ou após uma reinicialização. Neste modo, os processadores 80386/80486 se comportam de maneira semelhante a um 8086. Esta semelhança no comportamento é tal que um programa desenvolvido para o 8086 pode executar nos processadores 80386/80486 sem que perceba que está em outro processador.

O segundo modo é o modo protegido. Operando neste modo, os processadores 80386/80486 podem usar sua capacidade de endereçamento em 32 bits para acessar até 4 Gbytes de memória contínua e pode usar mecanismos internos de proteção e de paginação.

Os processadores 80386/80486 possuem também um modo de compatibilidade, entre o modo protegido e o modo real, chamado de modo virtual 8086, no qual mantém escondidas, porém ativas, as características próprias do modo protegido como a proteção e a paginação, enquanto executa código como se estivesse no modo real. Neste caso, executa código desenvolvido para o processador 8086 e esse código não percebe que está executando em uma variação do modo protegido. Esse modo de compatibilidade não deixa de ser também uma espécie de 'micro-kernel' pois, apesar de estar no modo protegido, executa endereçamento de maneira diferente da maneira normal de endereçamento deste modo.

Apesar dos processadores 80386/80486 poderem ser usados em sistemas de tarefa única, muitas facilidades de sua arquitetura foram projetadas para dar suporte à execução concorrente de múltiplas tarefas. Os processadores 80386/80486 podem executar uma troca de tarefa sob direção do sistema operacional ou automaticamente em resposta a uma interrupção ou exceção. A arquitetura dos processadores 80386/80486 define várias estruturas e registros, voltados ao tratamento de interrupções e gerenciamento de memória e multitarefa [14][15][16] [17] [25].

A seguir, veremos com detalhes esses modos de operação ressaltando as características que mais interessam para nosso desenvolvimento.

3.1 Modo Protegido

Como mencionamos anteriormente, operando neste modo, os processadores 80386/80486 pode usar sua capacidade de endereçamento em 32 bits para acessar até 4 Gbytes de memória contínua e pode usar mecanismos internos de proteção e de paginação. Nas seções a seguir iremos detalhar estes mecanismos e a forma de endereçar deste modo.

3.1.1 Endereçamento

Independente do modo de operação, os processadores 80386/80486 possuem três espaços distintos de endereçamento, cada um com sua área de abrangência e utilização. Na Fig. 3.1 ilustramos estas formas de endereçamento que são:

- **Espaço de endereçamento lógico** - utilizado pelos programas aplicativos e constituído de uma referência a segmento e um deslocamento no interior do mesmo. No modo real constitui-se de 16 bits para referência de segmento e de 16 bits para referência de deslocamento. No modo protegido, a referência de segmento permanece em 16 bits, porém a referência de deslocamento passa para 32 bits.
- **Espaço de endereçamento linear** - utilizado pelo processador após ter resolvido o cálculo entre as duas referências do espaço de endereçamento lógico. Constitui-se de um endereço de 32 bits, não importando o modo de operação do processador.
- **Espaço de endereçamento físico** - utilizado pelo processador para acessar a memória real. É constituido de 32 bits e é derivado do endereçamento linear através de uma tradução pelo mecanismo de paginação, desde que a mesma esteja habilitada. Essa tradução independe dos aplicativos e é efetuada internamente pelo processador, usando como base o endereço linear e as estruturas de dados internas da paginação. Se a paginação não estiver habilitada, o endereçamento físico é tomado como sendo igual ao endereçamento linear.

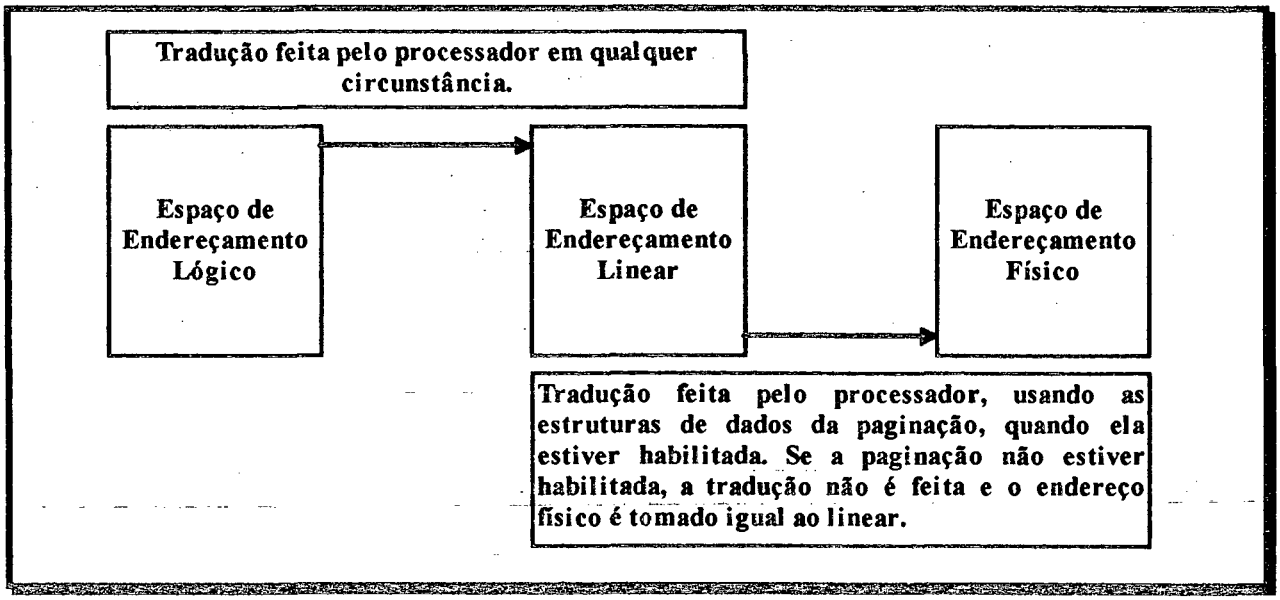

Fig. 3.1 - Espaços de endereçamento

Neste ítem nos prenderemos mais nas características do espaço de endereçamento lógico. As características do espaço de endereçamento linear veremos mais adiante, no decorrer do trabalho, quando tratarmos de paginação.

O espaço de endereçamento lógico dos processadores 80386/80486, no modo protegido, é inerentemente segmentado, porém existe uma grande flexibilidade na definição dos segmentos que é garantida por dois atributos:

- Os segmentos no modo protegido podem ter tamanhos variáveis desde um byte até quatro Gbytes.
- Os segmentos no modo protegido podem se sobrepor no espaço de endereçamento linear.

Um sistema operacional pode ser orientado a segmento, definindo até um máximo de 16384 segmentos de tamanho pequeno (porque 8192 é o tamanho máximo de cada uma das duas tabelas de descritores) ou pode ser indiferente à segmentação definindo poucos segmentos de tamanho grande. No limite, pode chegar até o ponto de definir segmentos com tamanho de 4 Gbytes que se sobreponham na memória de maneira a praticamente anular a segmentação.

Em qualquer dos casos, porém, uma tarefa ou programa deve ter um segmento de código (um seletor em CS) e um segmento de dados (seletor em DS) que podem ter o mesmo valor. Além destes, é obrigatório um segmento de pilha que pode ser separado ou pode usar o mesmo segmento de dados (o seletor em SS define a pilha corrente). Outros segmentos não são necessários, porém as instruções específicas de strings supõem um seletor válido em ES, o qual pode ser o mesmo que estiver em DS.

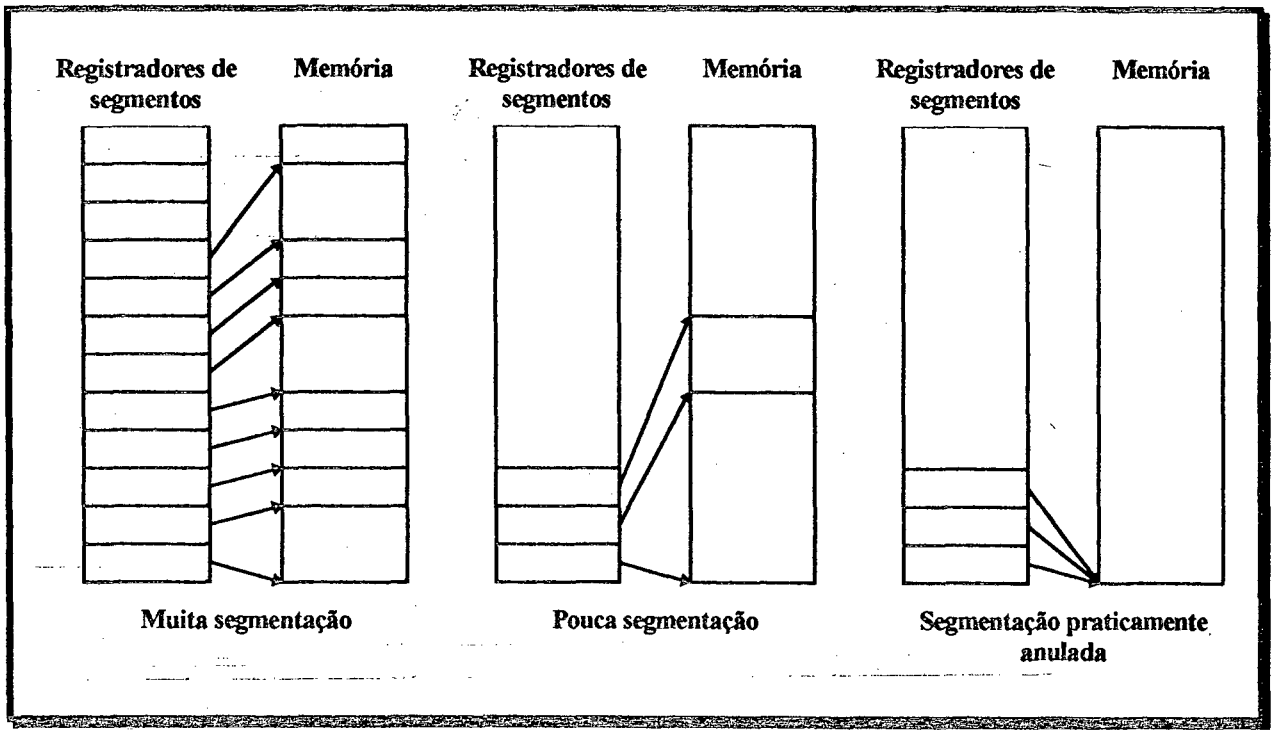

Fig. 3.2 - Uso da segmentação

Devido às restrições acima citadas e que decorrem do seu projeto, sempre teremos que usar segmentos ao endereçar nos processadores 80386/80486. É possível, porém, quase anular a existência de segmentos definindo-os como sobrepostos e do mesmo tamanho. Teremos neste caso os registradores todos com a mesma referência a segmento simulando um espaço de endereçamento não segmentado. Com tal procedimento, perderemos a possibilidade de empregar características de proteção entre estes segmentos, conforme veremos mais adiante, ao tratarmos de proteção.

3.1.2 Segmentação

Conforme vimos acima, os segmentos são a base do sistema de endereçamento dos processsadores 80386/80486 e podem ter qualquer tamanho. Esses tamanhos, se situados na faixa desde um byte até um megabyte, podem ser definidos com granularidade de um byte. Para definir tamanhos de segmentos maiores que um megabyte, , a granularidade aumenta para quatro quilobytes forçando a que, na faixa de quatro quilobytes até quatro gigabytes, os tamanhos de segmentos tenham que ser múltiplos deste valor.

Com essa flexibilidade de tamanho e com auxílio de compiladores adequados, o programador pode mapear os segmentos de maneira a se ajustarem a rotinas ou estruturas de dados.

Após decidir qual o modelo de segmentação que melhor se adapta aos objetivos de proteção e performance pretendidos para o sistema operacional, o projetista deve expressar o modelo através dos conteúdos das tabelas de descritores dos processadores 80386/80486.

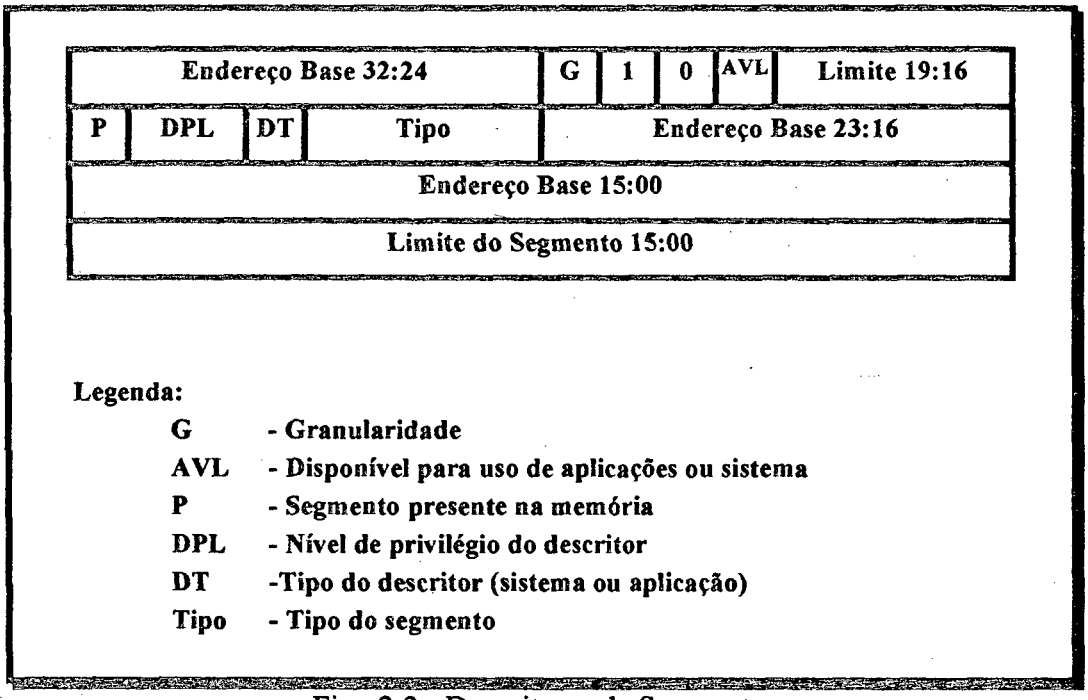

Fig. 3.3 - Descritores de Segmentos

No modo protegido, os segmentos dos processadores 80386/80486 são definidos pelos descritores de segmentos que estabelecem sua posição no espaço de endereçamento linear (endereço base), seu tamanho (limite) e seus atributos de proteção. A Fig. 3.3 mostra um esquema dos descritores.

Estes descritores estão organizados em duas estruturas de dados que são empregadas pelo processador durante o processo de endereçamento.

- **Tabela Geral de Descritores** (Global Descriptor Table - GDT), estrutura principal, comum a todas as tarefas.
- **Tabela Local de Descritores** (Local Descriptor Table - LDT), de uso exclusivo de uma tarefa ou de um grupo que a compartilhe, não podendo ser acessada por qualquer outra que não tenha sido criada explicitamente com permissão de acesso à mesma.

A organização do esquema de endereçamento prevê a possibilidade de se ter uma LDT para cada tarefa e ainda uma GDT comum para todas as tarefas do sistema. Uma tarefa pode ter seletores de segmentos na GDT e na LDT ao mesmo tempo.

É o sistema operacional quem cria os descritores durante a inicialização, porém eles são para uso do processador o qual os interpreta e atualiza. Como os processadores 80386/80486 não podem gerar um endereço linear para um segmento que não esteja coberto por um descritor, a distribuição dos descritores entre tarefas já dá um primeiro nível de proteção de acessibilidade de endereços lineares. O outro nível é dado pelos atributos de proteção dos descritores de segmento das tarefas.

O espaço de endereçamento lógico de uma tarefa é dado pelos descritores de segmento em duas tabelas de descritores, a tabela global de descritores (GDT), que é um recurso disponível a todas as tarefas e a tabela local de descritores (LDT), que é um recurso privativo da tarefa ou de um grupo de tarefas que a compartilhem. Essas tabelas são variáveis em tamanho tendo até o máximo de 64 quilobytes, o que limita suas capacidades em 8192 descritores cada uma delas. Os descritores de segmento existentes na GDT e na LDT da tarefa definem completamente o espaço de endereçamento linear que a tarefa pode gerar. Mesmo podendo gerar os endereços, não está garantida a acessibilidade, por parte da tarefa, a todos os segmentos, devido aos atributos de proteção dos descritores dos segmento.

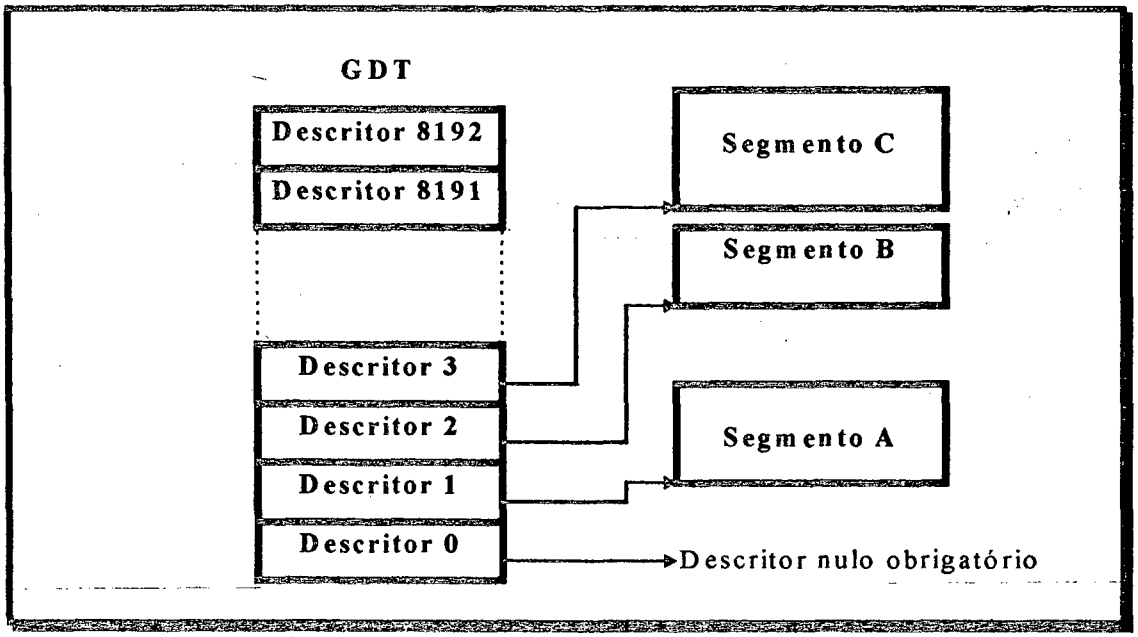

Fig. 3.4 - Organização de segmentos na GDT

Os registradores do sistema GDTR e LDTR apontam para as tabelas global e local de descritores, respectivamente. O registrador GDTR contém o endereço linear de 32 bits para a base da GDT e um limite de 16 bits. Na inicialização, o sistema operacional carrega o registrador GDTR com a instrução LGDT e, apesar de ser possível, normalmente não há motivo para alterar o valor carregado.

O sistema operacional também deve carregar o LDTR com um seletor, residente na GDT, que descreva o segmento onde está localizada a LDT da tarefa corrente, usando a intrução LLDT ou uma troca de tarefa simulada. Em cada troca de tarefa, o 80386/486 recarrega o LDTR com o seletor indicado no campo LDT do Segmento de Estado da Tarefa (Task State Segment -TSS) da nova tarefa.

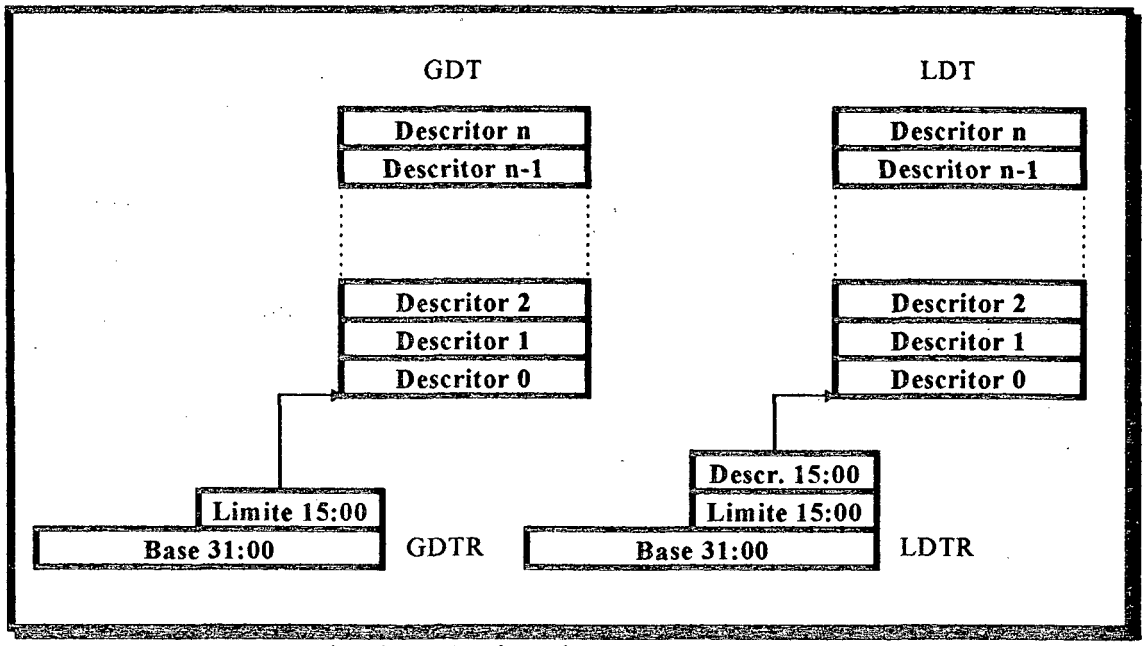

Fig. 3.5 - Registradores GDTR e LDTR

Um conceito importante com relação a segmentos é o de aliases. Diz-se que um segmento é alias de um outro quando os dois tem descritores diferentes e se referem ao mesmo espaço de endereçamento linear. Isso permite que uma tarefa tenha uma visão de um determinado segmento e o sistema operacional outra visão diferente.

É necessário, em certas condições, que se tenha essas maneiras distintas de ver um mesmo segmento, como por exemplo o caso de TSSs, cujos seletores não podem ser carregados em registradores de dados. Se for necessário ao sistema operacional alterar o conteúdo destes segmentos, é obrigatório o uso de aliases, um deles com atributos de segmentos de dados e permissão de escrita e o outro com atributos normais de TSS.

Na Fig. 3.6 ilustramos dois seletores apontando para o mesmo segmento, cada um deles com atributos diferentes, o que configura os dois como aliases.

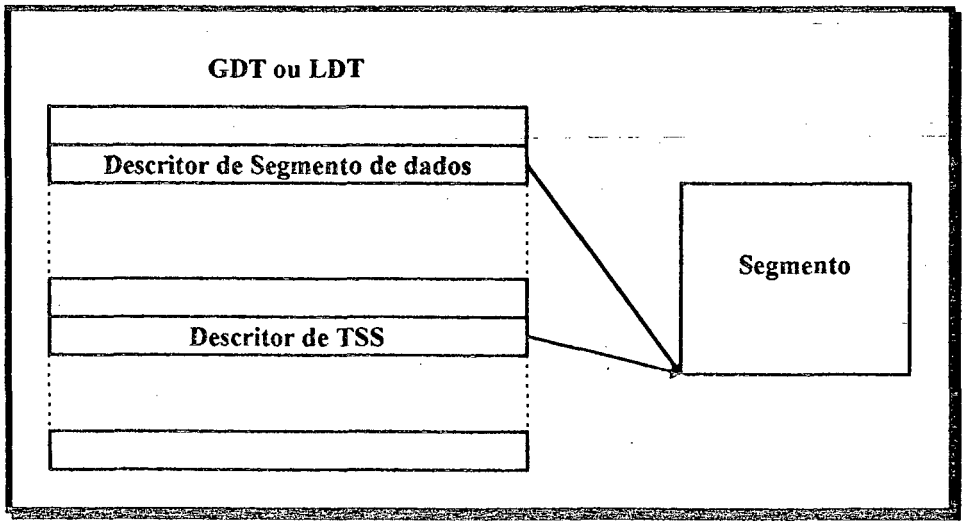

Fig. 3.6 - Segmentos aliases

3.1.3 Paginação

Todo o sistema operacional para os processadores 80386/80486 implementam obrigatoriamente algum tipo de segmentação, porém o uso da paginação é opcional [14][15][16] [17][20]. A paginação é usada tipicamente para implementar memória virtual mas suas facilidades de relocação de endereços e proteção podem ser usadas para outros fins. Como exemplo, imagine um sistema operacional que execute várias tarefas virtuais 8086 as quais geram internamente endereços lógicos no primeiro megabyte e que os endereços físicos correspondentes a estes endereços lógicos são relocados para qualquer outro local da memória. Essa relocação é feita durante a tradução do endereço linear para endereço físico, se o mecanismo de paginação estiver habilitado as estruturas de dados da paginação definirão a posição da memória física que será usada por aquele endereço lógico. Um sistema operacional habilita a paginação ligando o bit PG (paging - bit 31) no registrador CR0 através da instrução privilegiada MOV CR0 e desabilita a paginação zerando-o.

A paginação é implementada nos processadores 80386/80486 por baixo da segmentação, executando a tradução e a proteção de página logicamente após a tradução e a proteção de segmento. Por isso, é necessário muito cuidado na coordenação de proteções entre segmentos e páginas. Na verdade os procedimentos de tradução de endereços e de verificação da proteção de páginas e segmentos ocorrem concomitantemente, porém, para fins de projeto de sistemas, pode-se imaginar que os procedimentos derivados da paginação ocorram após os procedimentos de segmentação.

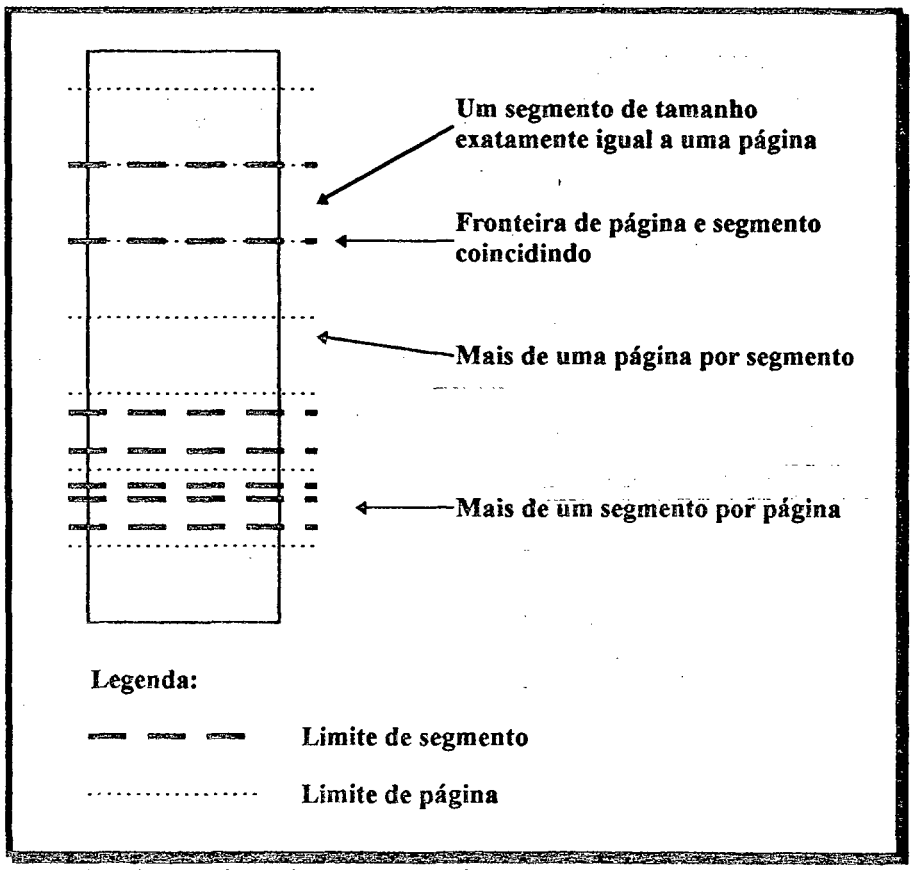

Fig. 3.7 - Disposições possíveis entre segmentos e páginas.

As páginas de memória tem tamanho fixo de quatro quilobytes e o mapeamento de segmentos e páginas podem ser quaisquer. Como os segmentos têm tamanho variável e

dependentes do modelo implantado pelo projetista do sistema, podemos ter uma página com vários segmentos dentro dela, podemos ter um grande segmento com várias páginas e, ainda, podemos ter limites de segmentos e páginas coincidindo. Qualquer distribuição entre segmentos e páginas é possível, conforme está ilustrado na figura a seguir.

A paginação é definida pelas estruturas internas que controlam a relocação de endereços. Essas estruturas são:

- Tabela de páginas
- Diretório de páginas

A paginação está baseada em estruturas de dados chamadas de Diretório de Páginas (PD - Page Directory), Tabelas de Páginas (PT - Page Table) e em um registrador chamado de Registrador Base do Diretorio de Páginas (PDBR - Page Directory Base Register).

Uma tabela de páginas do 80386/486 define uma coleção de páginas de 4 quilobytes de forma semelhante àquela com que uma tabela de descritores define um conjunto de segmentos de tamanho variável. Uma tabela de páginas é uma estrutura de dados composta por um conjunto de entradas de tabela de páginas (PTE - Page Table Entry). Uma tabela de páginas tem as seguintes características:

- Uma tabela de páginas tem tamanho igual ao tamanho de uma página, ou seja, quatro quilobytes.
- Uma tabela de páginas deve ter seu endereço base alinhado no endereçamento linear em 4 Kb, ou seja seus doze bits de ordem mais baixa devem ser iguais a zero.
- Uma tabela de páginas contém 1024 entradas cada uma delas definindo uma página de memória física de 4 quilobytes.
- Uma tabela de páginas, portanto, pode cobrir 4 megabytes de endereçamento linear ou físico.

Com uso tipicamente para implementar memória virtual, uma entrada de tabela de página que não esteja presente na memória tem seu formato dependente do sistema operacional. Uma entrada de tabela de páginas presente na memória possui informações de endereçamento, de proteção e de memória virtual. O endereço base consta dos 20 bits superiores do endereço físico com os 12 bits de menor ordem iguais a zero, por ser cada página alinhada em 4 Kb. Os dados de proteção são os bits de leitura /escrita e de usuário/supervisor que protegem contra acessos não autorizados ou função inadequada. Os bits para memória virtual são A (accessed) e D (dirty) que mostram se a página foi acessada recentemente e se sofreu operação de escrita desde que foi carregada do disco.

Na Fig. 3.8, mostramos o conteúdo das entradas de tabelas de páginas.

O endereçamento linear que uma tarefa desenvolvida para os processadores 80386/80486 pode gerar está definida nos descritores de segmento da tarefa e, no caso geral, pode ser qualquer endereço dentro do espaço linear de 4 gigabytes.

Quando a paginação está habilitada, os processadores 80386/80486 precisam verificar todos os endereços lineares utilizados pelo aplicativo que está sendo executado através de consulta a uma tabela de páginas para saber se os endereços são válidos. Para cobrir todo o

espaço de endereçamento linear seriam necessárias 1024 tabelas de páginas para cada uma das tarefas.

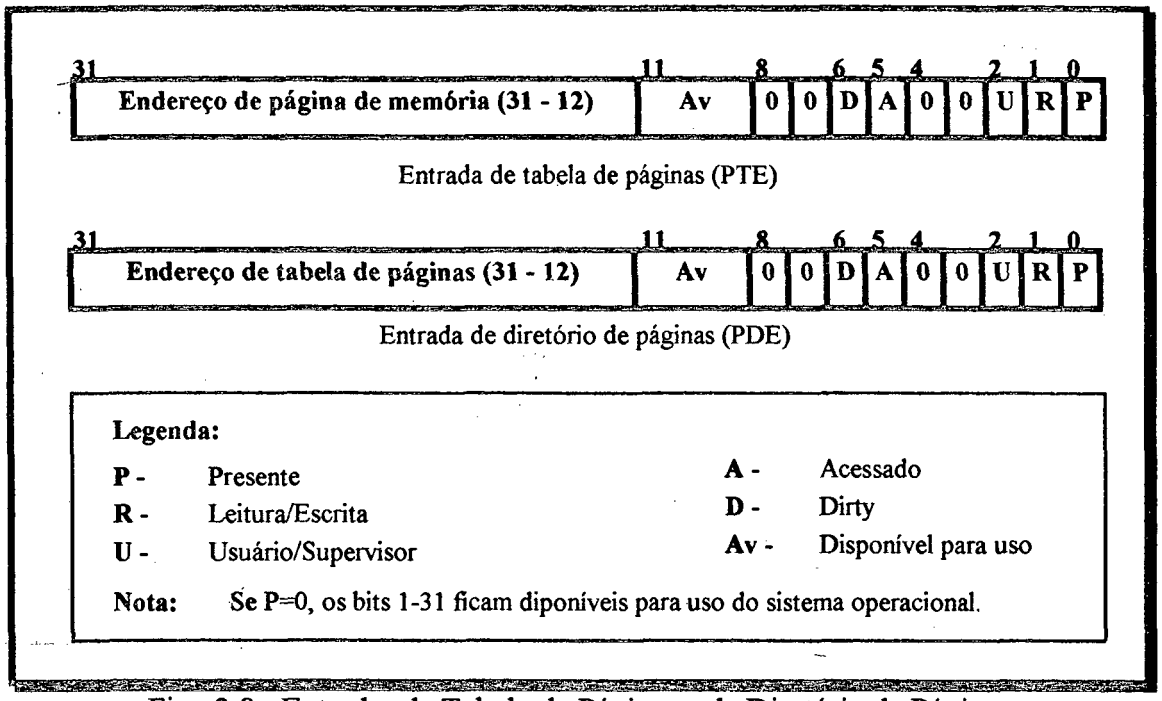

Fig. 3.8 - Entradas de Tabela de Páginas e de Diretório de Páginas

Para não forçar o uso excessivo de memória, alocando a quantidade necessária para um conjunto completo de 1024 tabelas de páginas, a maioria das quais não estaria presente, pois nenhum microcomputador tem capacidade de instalar quatro gigabytes de memória, os processadores 80386/80486 tem uma tabela de páginas de ordem mais elevada chamada de diretório de páginas.

Um diretório de páginas é muito semelhante a uma tabela de páginas, pois tem tamanho também igual a uma página e também deve estar alinhado em endereços físicos múltiplos de 4 quilobytes. Cada entrada no diretório de páginas (PDE) define endereço e atributos de uma tabela de páginas. Assim, as páginas tem dois conjuntos de atributos:

- Um primeiro conjunto que está definido na entrada do diretório de páginas. Seus atributos tem efeito sobre todas as entradas da tabela de páginas correspondente.
- Um segundo conjunto que está definido na entrada da tabela de páginas. Seus atributos tem efeito só para a página de memória física correspondente. Esse conjunto controla, então, uma página de memória física, ou seja, quatro quilobytes de memória.

De acordo com o que vimos, uma entrada de diretório de páginas define um conjunto de atributos que se aplicam a 1024 entradas na tabela de páginas, cada uma referente a uma página de 4 Kbytes o que, no total, corresponde a 4 Mbytes de espaçamento linear. Marcando-se uma entrada no diretório de páginas como não presente tem o mesmo efeito de se marcar todas as entradas de uma tabela de páginas como não presente. Esse procedimento traz como benefício a possibilidade de economizar memória porque não é necessário alocar memória para as tabelas de páginas que não estejam presentes na memória.

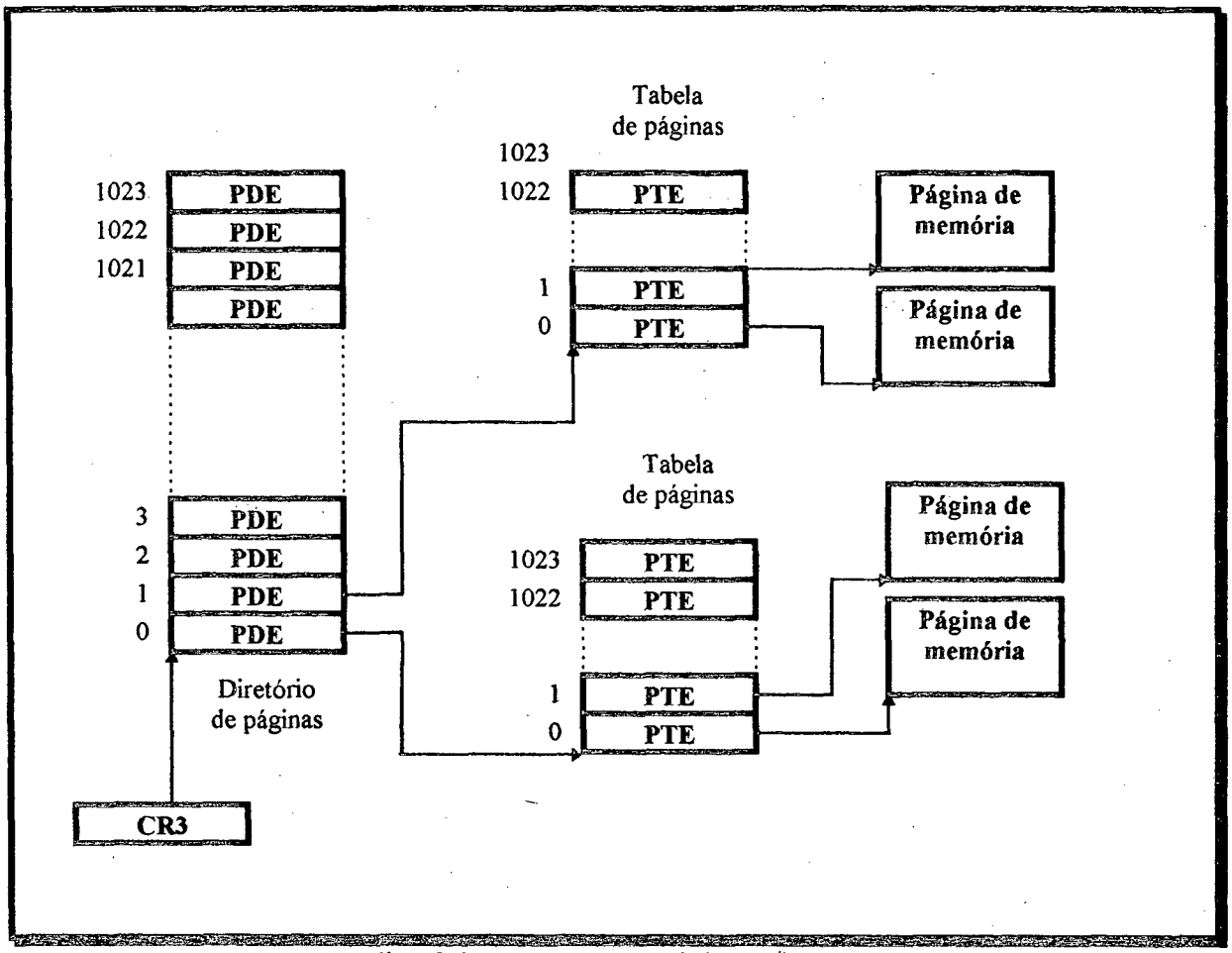

Fig. 3.9 - Esquema geral da paginação

Como exemplo, vejamos um computador com dezesseis megabytes de memória. Para habilitar a paginação precisariamos de um diretório de páginas com quatro entradas marcadas como presentes apontando para quatro tabelas de páginas. As demais entradas do diretório de páginas seriam marcadas como não presentes. Usaríamos, então, cinco páginas de memória para o diretório de páginas e as quatro tabelas de páginas, gastando vinte quilobytes de memória em alocações de estruturas. Se não fosse assim, se tivéssemos que alocar tabelas de páginas para entradas no diretório de páginas marcadas como não presentes, seriam necessárias 1025 páginas para acomodar o diretório de páginas e as 1024 páginas correspondentes, ocupando um total de mais de quatro megabytes de memória.

O registrador do sistema CR3 contém o endereço físico do diretório de páginas corrente. Durante a inicialização, o sistema operacional pode carregar o registrador CR3 usando a instrução privilegiada MOV CR3. Durante uma troca de tarefa, o registrador CR3 é carregado automaticamente pelo processador usando o valor armazenado no campo CR3 do TSS da nova tarefa, se este valor for diferente do valor corrente. A Fig. 3.9 mostra como se organizam as estruturas de dados da paginação.

Duas entradas nas tabelas de páginas que contenham o mesmo endereço físico são ditas aliases, da mesma maneira que os segmentos. O mesmo acontece com entradas no diretório de páginas. Os aliases podem ter atributos diferentes para uso por parte de tarefas e por parte do sistema operacional.

Duas ou mais tarefas podem partilhar todas as suas páginas compartilhando o mesmo diretório de páginas. Essa abordagem é adequada para sistemas operacionais que estejam usando paginação apenas para implementar algum mecanismo de memória virtual. Tarefas com diretórios de páginas separados podem compartilhar tanto tabelas de páginas como páginas de memória individuais. Duas tarefas podem compartilhar uma tabela de páginas se ela for alias em seus respectivos diretórios de páginas e podem compartilhar uma página se esta for alias em uma de suas respectivas tabelas de página.

Como já foi mencionado, os processadores 80386/80486 testam a proteção de segmento antes de testar a proteção de página. Como já foi mencionado, se for usado o atributo na entrada do diretório de páginas, ele se tornará válido para todas as tabelas de páginas desta entrada no diretório de páginas. Os atributos de proteção de página são:

- Bit U/S (user/supervisor) que, quando é igual a um (usuário), permite qualquer tarefa acessar a página e, quando é igual a zero, permite acesso apenas a tarefas executando nos níveis 0, 1 ou 2 (supervisor).
- Bit R/W (read/write) que controla o acesso para escrita das tarefas que estiverem executando no nível de privilégio 3 (usuário). Se o bit R/W for igual a zero, essas tarefas só poderão ler e executar a página. Tarefas mais privilegiadas podem sempre ler, escrever e executar quaisquer páginas.
- Todas as entradas possuem três bits disponíveis (AV) para guardar atributos de páginas definidos pelo sistema operacional que não são alterados ou interpretados pelo processador.

A tradução de um endereço linear em endereço físico é realizada pelos processadores 80386/80486 procurando uma entrada de diretório de página e usando esta entrada para identificar uma tabela de páginas. Nesta tabela de páginas os processadores 80386/80486 seguem procurando uma entrada que, então irá identificar a página física da memória que foi endereçada.

Essa tradução, que toma algum tempo, é acelerada usando um buffer interno ao chip chamado "Translation Lookaside Buffer". Esse mecanismo não passa de um cache das entradas do diretório de páginas e das entradas de tabelas de páginas mais recentemente usadas. O manuseio deste buffer é de grande importância para sistemas que tratem de memória virtual, no tratamento das informações que o processador automaticamente atualiza durante o uso do mesmo.

3.1.4 Proteção

Como os processadores 80386/80486 não podem gerar um endereço linear para um segmento que não esteja coberto por um descritor, a distribuição dos descritores entre tarefas já dá um primeiro nível de proteção de acessibilidade de endereços lineares. Outro nível de proteção é dado pelos atributos de proteção dos descritores de segmentos que as tarefas podem acessar. Além dos testes normais de permissão de leitura e escrita, os processadores 80386/8080486 podem testar os acessos considerando tipos adequados de utilização das informações contidas nos segmentos (código ou dados), pode testar o tamanho dos segmentos e níveis de privilégio dos mesmos (quatro níveis).

Todos estes testes visam a segurança e proteção e ajudam a descobrir erros que não podem ser descobertos durante a compilação (índices de arranjos que ultrapassam os limites da estrutura de dados e ponteiros que endereçam segmentos errados).

A definição de segmentos que se ajustem perfeitamente a estruturas de dados ou a funções de código, permite que a proteção seja aplicada a nível de objetos definidos pelo programador.

A proteção de segmentos durante a execução toma tempo e os processadores 80386/8080486 reduzem esse tempo testando os atributos dos segmentos em paralelo com a tradução do endereçamento lógico para o endereçamento linear. Outros testes, como validade do seletor e do descritor, são efetuados apenas quando um seletor é carregado em um dos registradores de segmento.

Uma tarefa que use múltiplos segmentos de dados ou segmentos distintos para pilha e dados deve usar ponteiros de 48 bits (16 bits do seletor e 32 bits do offset) para indicar, sem ambigüidades, o segmento a que o ponteiro se refere. Comparados com ponteiros de 32 bits (apenas offset) os ponteiros segmentados consomem mais memória e usam um ciclo de barramento a mais para transferir dados para ou da memória. Esse fator deve ser levado em consideração ao se decidir entre vários segmentos ajustados às estruturas de dados e partes de código com muita proteção e segmentos maiores e em menor número e menor grau de proteção.

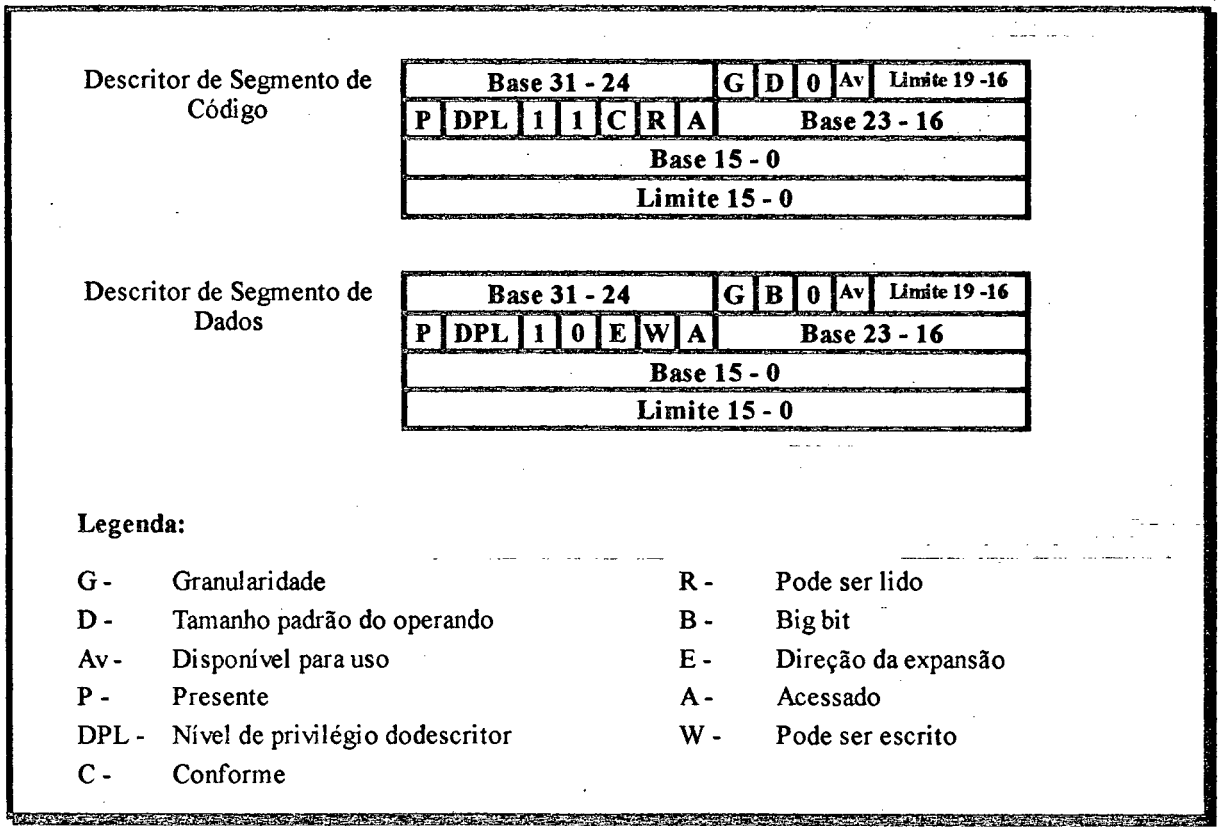

Fig. 3.10 - Descritores dos segmentos de código e dados

Os campos de proteção dos descritores permitem ao sistema operacional definir as condições sob as quais os segmentos associados podem ser acessados. Se uma tentativa viola uma destas condições, os processadores 80386/80486 não executam o acesso e, em seu lugar, geram uma exceção. Os processadores 80386/80486 distinguem os segmentos de código e dados através do bit código/dados do descritor do segmento.

Uma tentativa para escrever em um segmento de código ou de transferir o controle da execução para um segmento de dados gera uma falha de proteção geral. Além disso, pode-se restringir ainda mais as operações que uma tarefa pode fazer em segmentos de código (podem ser lidos e executados) e dados (podem ser lidos e escritos) através do bit R (readable) de um segmento de código ou do bit W (writeable) de um segmento de dados. Se forem zerados, o segmento de código só pode ser executado e o segmento de dados só pode ser lido. Um segmento de código também pode ser classificado como 'conforming' indicando que não tem nível de privilégio definido, sendo executado no nível de quem chamou a rotina.

Para proteger contra acessos fora do espaço linear do segmento os processadores 80386/80486 comparam a parte do offset do endereço lógico com o limite do segmento. Se uma tarefa tem um erro que cause a geração de um endereço fora do segmento, os processadores 80386/80486 não executam o salto ou acesso e geram uma exceção de proteção geral.

O campo nível de privilégio do descritor (DPL) define o nível de privilégio do segmento, sendo zero o mais privilegiado e tres o de menor privilégio. Normalmente, uma tarefa tem seu nível de privilégio corrente (CPL) igual ao nível de privilégio do código que está executando. Apenas no caso em que esteja executando um segmento de código conforme, terá o nível de privilégio do segmento de código que tranferiu controle para o segmento conforme.

Quando o operando de uma instrução for um seletor, o nível de privilégio corrente da tarefa pode ser reduzido para a execução daquela tarefa, pelo campo nível de privilégio requisitado (RPL) do seletor. Quando o RPL for zero, não terá efeito no nível de privilégio corrente.

O nível de privilégio do segmento define o privilégio necessário para acessar o segmento. Para ler ou escrever em um segmento de dados uma tarefa deverá ser ao menos tão privilegiada quanto o segmento, ou seja, pode acessar dados em segmentos de nível de privilégio igual ou menor.

Para transferir a execução para outro segmento de código através das instruções JMP, CALL, RET ou IRET, uma tarefa deve ter o mesmo privilégio que o segmento destino. Para os casos especiais de chamadas do sistema, interrupções e exceções, nos quais o nível de privilégio da tarefa é aumentado enquanto executa um segmento de código mais privilegiado, o 80386/486 possui um descritor especial chamado 'gate'.

Não é permitido transferir a execução para segmentos de código menos privilegiado através da instrução CALL porque isso implicaria na possibilidade de se retornar para segmentos mais privilegiado através da instrução RET, violando os critérios de proteção.

O nível de privilégio de uma tarefa também define, além da acessibilidade, as instruções que uma tarefa pode executar. Instruções privilegiadas, como as que permitem alterar o 'flag' de interrupção, só podem ser executadas no nível zero. Para executar instruções de entrada/saida uma tarefa deve ter ao menos tanto privilégio quanto seu nível de privilégio de entrada e saida (IOPL, no registrador EFLAGS durante a execução da tarefa).

Além destas informações, nos descritores são disponíveis: um bit que o sistema operacional pode usar para seus próprios fins, um bit D (default operand size)que define o tamanho default do operando e de offsets e os bits P (present) e A (accessed) para auxiliar na implementação de memória virtual a nível de segmento.

3.1.5 Tratamento de Interrupções

Interrupções e exceções são eventos não programados que alteram o fluxo normal de execução através de instruções de uma tarefa.

Interrupções ocorrem independente de instruções e tipicamente avisam pedidos de serviços de dispositivos externos [13][14][15][16][17][29][30]. Os processadores 80386/80486 reconhecem uma interrupção entre a execução de uma instrução e outra e, no caso de instruções de strings, entre ciclos de repetição.

Diferentemente de interrupções, exceções são resultantes da execução de instruções, como ocorre quando encontra um erro em uma instrução. Por essa definição a execução de uma instrução INT *n* é, na verdade, uma exceção.

Cada tipo diferente de interrupção ou de exceção possui um número *n* entre 0 e 255 e requer um tipo diferente de resposta. O sistema operacional associa o número *n* com uma fonte de interrupção através da programação do controlador 8259A - Controlador Programável de Interrupção (PIC - Programable Interrupt Controler).

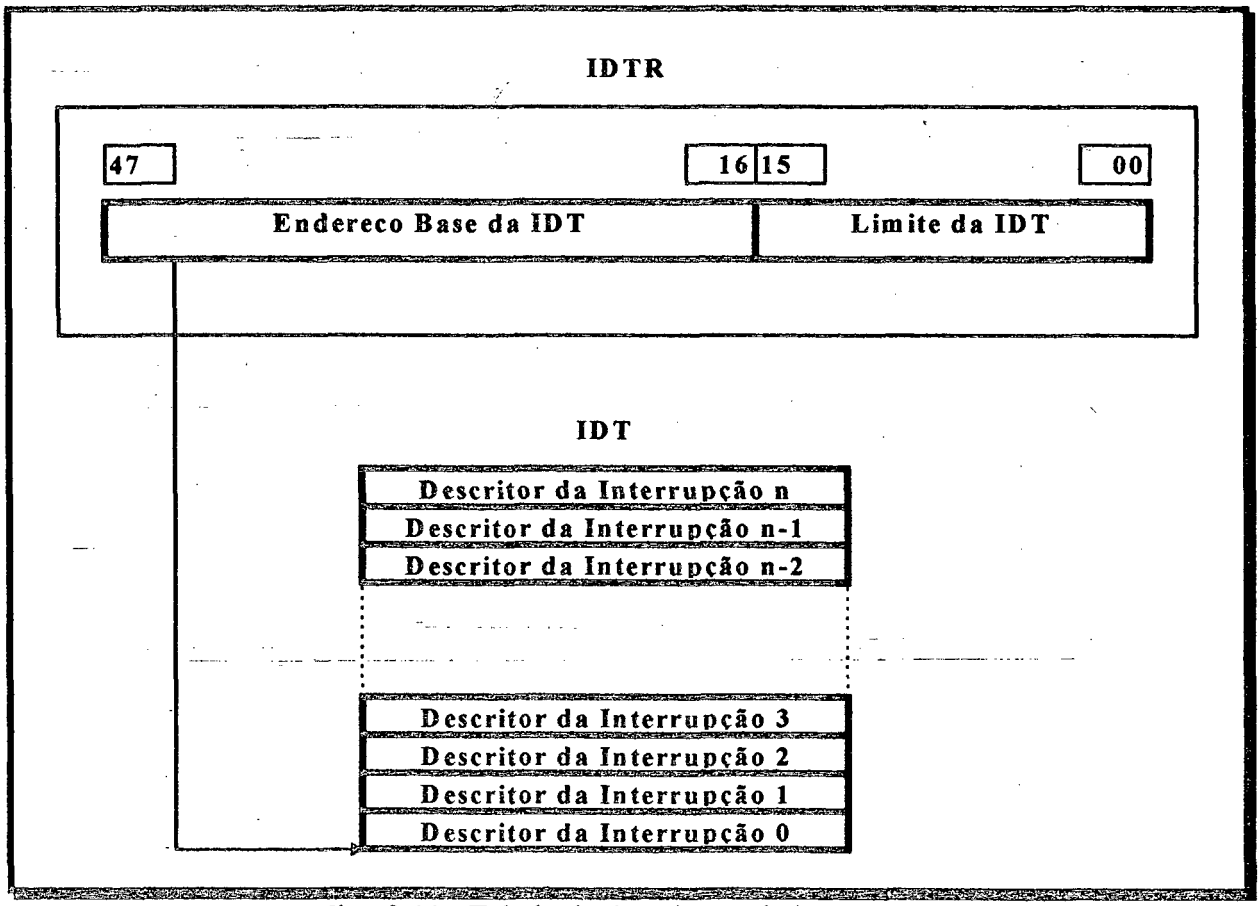

Fig. 3.11 - Tabela de descritores de interrupção

A Tabela de Descritores de Interrupção (Interrupt Descriptor Table - IDT) é o elo que o sistema operacional usa para ligar um número *n* de interrupção ou exceção com a rotina de manipulação designada para atender a este tipo de interrupção ou exceção.

O número *n* é usado como índice para a IDT e o descritor nesta tabela, identificado através deste índice, possui as informações necessárias para transferir o controle da execução para a rotina de manipulação.

Da mesma maneira que a as tabelas de descritores de segmentos que vimos anteriormente, GDT e LDT, a IDT também é uma tabela de descritores, só que descreve rotinas de manipulação de interrupção e tarefas de manipulação de interrupção. Durante a inicialização, o sistema operacional monta a IDT e carrega seu endereço físico no registrador de sistema IDTR, através do uso da instrução LIDT. A IDT pode estar localizada em qualquer local do espaço de endereçamento linear.

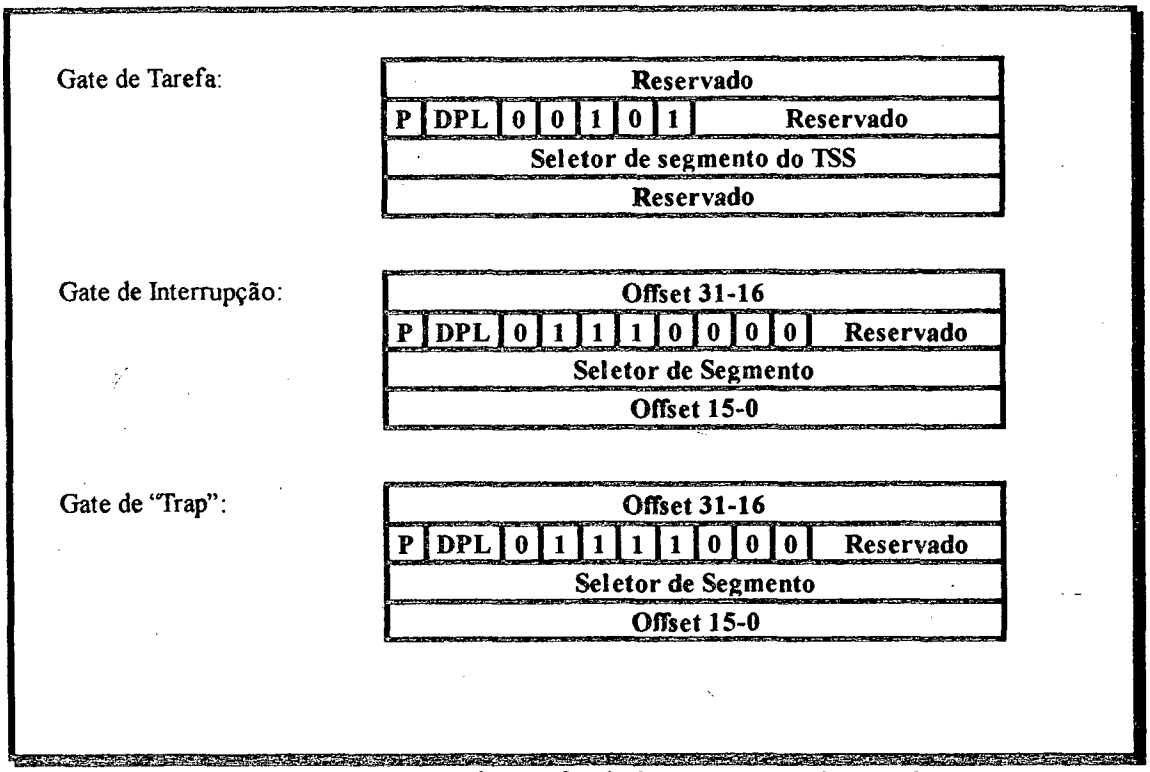

Fig. 3.12 - 'Gates' de tarefa, de interrupção e de 'trap'.

Os descritores colocados na IDT podem ser de três tipos diferentes de acordo com seu comportamento no tratamento da interrupção:

- Um gate de interrupção contém um seletor de segmento e um deslocamento para endereçar a rotina de manipulação que tratará da interrupção ou da exceção. Esse tratamento será efetuado dentro do ambiente da tarefa corrente e o manipulador será chamado com as interrupções desabilitadas.
- Um gate de "trap" também contém um seletor de segmento e um deslocamento para endereçar a rotina de manipulação que tratará da interrupção ou da exceção. Ainda nesse caso, o tratamento será efetuado dentro do ambiente da tarefa corrente, porém o manipulador será chamado sem alterar o estado das interrupções.
- Um gate de tarefa contém o seletor de um TSS representando a tarefa que deverá manipular a interrupção ou exceção. O tratamento da interrupção se dará através de uma troca de tarefa, ocorrendo dentro do ambiente da tarefa de manipulação da interrupção.

Dependendo da forma como se coloca o descritor na IDT, um manipulador de interrupção ou de exceção pode ser implementado tanto como uma tarefa que roda em seu próprio ambiente ou como uma rotina que roda no ambiente da tarefa em que a interrupção ou exceção foi acionada. Cada forma de implementação tem seus méritos e deve ser cuidadosamente escolhida em função das necessidades, como, por exemplo:

- Acesso a dados no contexto em que a interrupção ocorreu.
- Independência de ambiente.
- Rapidez no atendimento.

Só um exame detalhado do caso poderá definir que tipo de manipulador de interrupção melhor se adaptará. Rotinas de interrupção cumprem o primeiro e o último ítem, porém o segundo só pode ser cumprido pelas tarefas de interrupção, que tem como desvantagem a troca de tarefa lenta pela necessidade de trocar também o contexto.

Um manipulador de interrupção roda normalmente no nível de privilégio zero, porém, no caso geral, o manipulador deve ser, ao menos, tão privilegiado como o código em que pode ocorrer a interrupção. Isso se deve ao fato dos processadores 80386/80486 não chamarem jamais um código de privilégio inferior, fato que é mantido mesmo no tratamento de interrupções e exceções.

3.2 Modo Real

O modo real é o modo nativo dos processadores 8086/8088 e foi mantido nos processadores mais modernos por motivo de compatibilidade, perseguida durante toda a evolucão da família de processadores da Intel. Após uma reinicialização, os processadores 80386/80486 iniciam a operação no modo real. Se o sistema operacional o desejar, pode comutar para o modo protegido ou para o modo virtual 8086. Tal procedimento se justifica porque o sistema operacional mais largamente utilizado é o DOS e seus compatíveis. O DOS foi desenvolvido especialmente para os processadores 8086/8088, por encomenda da IBM quando criou a linha de computadores pessoais chamada de IBM PC-XT. Iniciando a operação no modo real, os processadores 80386/80486 entram no modo real e permitem a utilização do DOS, funcionando, então, como se fossem processadores 8086/8088 de grande velocidade.

3.2.1 Endereçamento

O espaço de endereçamento linear do modo real também é segmentado, porém sem a flexibilidade existente no modo protegido. As características que determinam os segmentos no modo real são:

- Os segmentos tem tamanho fixo de 64 Kb.
- Os segmentos sempre iniciam em endereços múltiplos de 16 bytes.
- Tanto o endereço inicial do segmento quanto os deslocamentos em seu interior são compostos por 16 bits.
- Os segmentos podem se sobrepor no espaço de endereçamento linear.

De acordo com estas características, vemos que a perda de flexibilidade se dá na definição do tamanho dos segmentos e no seu endereço inicial.

Ainda aqui devemos ter, para qualquer programa, segmentos de dados, código e pilha nos registradores adequados do processador, porém sua organização pode ser qualquer dependendo do modelo de código.

3.2.2 Segmentação

No modo real, os segmentos ainda são a base do endereçamento dos processadores 80386/80486. Eles tem tamanho fixo de 64 Kb e endereços iniciais múltiplos de 16 bytes. O projetista do sistema operacional deve escolher o modelo de segmentação pretendido para o mesmo e cuidar por si só dos mesmos pois, neste modo, não existe nenhuma estrutura de dados do sistema para gerenciar estas informações, como a GDT no modo protegido.

Além disso não existe qualquer esquema de proteção para estes segmentos, sendo possível que uma tarefa acesse código do sistema operacional ou dados do mesmo ou de outras tarefas. O funcionamento de todo o sistema depende do bom comportamento de todos. Isso decorre da origem do DOS, sistema operacional original, ser monotarefa. Assim, os problemas de conflito ocoreriam apenas entre uma tarefa e o sistema operacional, não prejudicando ninguém mais.

A tradução dos endereços lógicos para endereços lineares no modo real usa os 20 bits que foram definidos inicialmente para os processadores 8086/8088. Os endereços são gerados tomando os 16 bits do endereço inicial do segmento e completando-os com 4 bits zerados à sua direita e somando com os 16 bits do deslocamento completado com 4 bits zerados à sua esquerda.

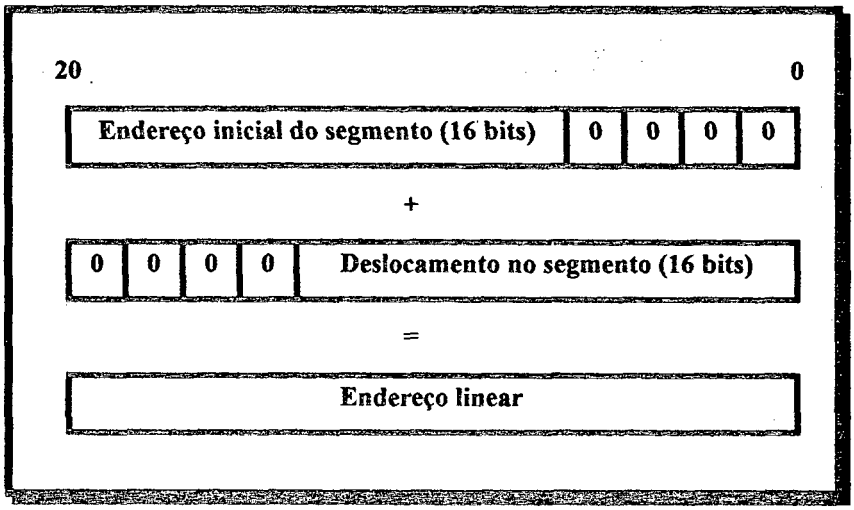

Fig. 3.13 - Tradução de endereços no modo real

3.2.3 Paginação

No modo real não temos acesso à paginação. Apesar dos mecanismos existirem no processador, o acesso não é possível neste modo.

3.2.4 Proteção

Da mesma forma que a paginação, não temos acesso aos mecanismos de proteção existentes no processador, quando operamos no modo real.

3.2.5 Tratamento de Interrupções

No modo real, as interrupções e exceções são interpretadas da mesma forma que no modo protegido. A diferença fica por conta dos manipuladores, cujos endereços estão colocados em uma posição fixa da memória, bem no início, e ocupam 4 bytes cada um (2 para segmento e 2 para deslocamento). Todos os manipuladores são rotinas de interrupção e podem estar em qualquer local dentro do espaço endereçável da memória. Tal agrupamento de endereços é conhecido como 'vetor de interrupção'. Os manipuladores de interrupções são executados no mesmo contexto do código que foi interrompido e chamados com as interrupções desabilitadas. Isso não impede que os mesmos usem pilhas e dados próprios desde que troquem o conteúdo dos registradores adequados. Também podem executar com as interrupções desabilitadas, desde que as desabilitem explicitamente.

3.3 Modo Virtual 86

Descrevemos, anteriormente, os dois modos de principais de operação dos processadores 80386/80486. Cada um deles tem características próprias e bastante distintas. No primeiro deles destacamos endereçamento amplo, facilidades de projetar as aplicações em termos de segmentos, proteção, facilidades para multitarefa e paginação. No segundo, segmentos de tamanho fixo, endereçamento restrito, falta de proteção, ausência de paginação e falta de suporte à multitarefa.

No terceiro modo, modo virtual 8086, que passaremos a descrever, a principal característica é a compatibilidade entre os dois modos anteriores. Temos, neste modo, endereçamento compatível com o modo real e, por trás, sem que os aplicativos percebam, estão mantidas todas as características do modo protegido, como proteção e paginação.

Portanto, os processadores 80386/80486 podem executar código gerado para o 8086 em dois modos:

- no modo real e
- no modo virtual 8086 (V86).

No modo real os processadores 80386/80486 se comportam como um 8086 muito rápido porém não oferece facilidades para tarefas e para proteção, rodando uma única tarefa em um ambiente não protegido. No modo virtual 8086, que é um submodo do modo protegido, os processadores 80386/80486 fornecem um ambiente de tarefa protegido onde um programa para o 8086 pode executar sem interferir com o sistema operacional ou com outras tarefas.

A maior diferença entre o modo real e modo virtual 8086 está nos níveis de privilégio e na manipulação de interrupções e exceções, enquanto que a execução de instruções e endereçamento são bastante similares. Os atributos principais de uma tarefa V86 são:

- O bit VM em EFLAGS deve estar ligado.
- O nível de privilégio é sempre igual a 3 enquanto executando as instruções no modo virtual. Interrupções e exceções trocam o modo de operação do processador do modo virtual 8086 e nível de privilégio 3 para o modo protegido no nível de privilégio 0. Uma instrução IRET, após o tratamento da interrupção, o levará de volta ao modo V86 e nível de privilégio 3, no ponto de interrupção.

- Uma tarefa V86 pode executar concorrentemente com tarefas de modo protegido, com outras tarefas V86 e tarefas para o 80286.
- Tarefas V86 são compatíveis com paginação e memória virtual.
- A uma tarefa V86 pode ser dada permissão para se referir diretamente a dispositivos de entrada e saída e acessar o flag de habilitação de interrupção ou essas referências podem ser interceptadas, desviadas e simuladas pelo sistema operacional.

3.3.1 Endereçamento

No modo virtual, os endereços são traduzidos do espaço de endereçamento lógico para o espaço de endereçamento linear de forma idêntica à tradução feita no modo real.

3.3.2 Proteção

A proteção, como existente no modo protegido, não é aplicável entre segmentos no modo virtual. Para proteger o código do sistema operacional que estiver executando dentro do espaço de endereçamento lógico de uma tarefa virtual, deve-se usar os atributos de proteção do sistema de paginação. Para proteger uma tarefa de outra, deve-se reservar um espaço de endereçamento linear de um megabyte localizado a partir do endereço zero para cada tarefa e mapeá-los para posições diferentes no espaço de endereçamento físico. Como cada tarefa virtual não pode gerar endereços fora de um megabyte, não poderá interferir na outra.

3.3.3 Paginação

A paginação permite a execução concorrente de multiplas tarefas no modo virtual, permite proteção e isolamento do sistema operacional. Apesar de não ser necessário habilitar a paginação para executar tarefas no modo virtual, deveremos fazê-lo se desejarmos executar mais de uma delas ou relocar o espaço de endereçamento linear de tarefas no modo virtual para endereços físicos acima de um megabyte. O hardware de paginação permite que o espaço de endereçamento linear de vinte bits das tarefas virtuais seja dividido em 256 páginas. Cada uma delas pode estar localizada em qualquer lugar dentro do espaço de endereçamento de 32 bits dos processadores 80386/80486. Adicionalmente, como o registrador CR3 é carregado a cada troca de tarefa, cada tarefa virtual pode ter o seu próprio esquema de endereçamento, inclusive compartilhando o código do sistema operacional e pode, ainda , ter endereçamento linear maior que a memória física existente, criando a memória virtual.

3.3.4 Tratamento de Interrupções

Quando acontece uma interrupção no modo virtual 8086, o processador troca do modo virtual 8086 para o modo protegido antes de chamar o manipulador e este a trata da mesma maneira que vimos quando citamos o tratamento de interrupção no modo protegido. Após tratar a interrupção, quando o manipulador executa uma instrução IRET, os processadores 80386/80486 voltam para o modo V86 e continua a execução da tarefa virtual. Para que este mecanismo funcione adequadamente, é necessário que:

- Todos os 'gates' na IDT sejam 'gates' dos processadores 80386 ou 80486. Não são permitidos 'gates' do processador 80286.

- Manipuladores baseados em rotinas devem estar em segmentos de código de nível de privilégio zero e não conformes. Manipuladores baseados em tarefas podem rodar em qualquer nível de privilégio.

Quando uma tarefa V86 causa uma exceção ou é interrompida, o 80386/486 chama um manipulador de maneira a minimizar a diferença entre uma tarefa virtual 8086 e uma tarefa do modo protegido.

O objetivo deste procedimento é sistematizar o tratamento quando o sistema operacional gerenciar tarefas virtuais 8086 e protegidas ao mesmo tempo. Se o manipulador for uma tarefa, o 80386/486 troca de tarefa como usual, salvando o estado da máquina no TSS antigo. Se o manipulador for uma rotina, o 80386/486 primeiro salva os registros de segmento da tarefa V86 na pilha de nível de privilégio zero e, então, salva EFLAGS, endereço de retorno e código de erro, se a interrupção ou exceção precisar, conforme usual.

Como os elementos do topo da pilha são idênticos tanto se o manipulador for chamado no modo V86 como no modo protegido, o manipulador pode retornar sempre com uma instrução IRET sem se preocupar com qual modo estava antes de ser invocado. Assim, a presença de tarefas virtuais 8086 é transparente a manipuladores de interrupção ou exceção baseados em rotinas, apesar de algumas vezes ser necessário ao manipulador examinar o bit VM do registrador EFLAGS para saber em que modo foi invocado.

No modo virtual 8086, executamos programas que foram desenvolvidos para o DOS e, portanto, as tarefas deste modo geram exceções que o sistema operacional do 80386/486 deve manipular de maneira especial. Por exemplo, a simples instrução INT 21H, que é válida e bastante usada para o 8086, gera uma exceção se for executada no modo V86.

É conveniente juntar todas as rotinas que respondem à exceções do modo V86 em uma coleção normalmente chamada de Monitor Virtual. Um monitor virtual simula as instruções do 8086 que o 80386/486 não executa no modo V86.

Para simular uma instrução do 8086, o monitor virtual precisa localizar e decodificar a instrução, o que é possível através de CS e EIP colocados na pilha do manipulador. Depois de simular a instrução, se for mesmo uma instrução válida do 8086 e não uma instrução errada do 80386/486, o monitor virtual deve incrementar o valor de EIP na pilha para poder retornar para uma instrução após àquela instrução que estava sendo apontada e que já foi simulada.

Alguns programas do 8086 desabilitam interrupções enquanto executam operações críticas. Um sistema operacional 80386/486 pode permitir que uma tarefa V86 altere o flag IF ou o sistema operacional pode ser instruído a gerar uma exceção de falha de proteção geral se a tarefa tentar carregar ou armazenar IF.

Uma tarefa V86 executa no nível de privilégio 3 e sua habilitação em alterar IF é definida por seu IOPL. Se o IOPL da tarefa for inferior a 3, uma exceção será gerada se tentar alterar IF. Se IOPL for 3, o 80386/486 executará todas as instruções relacionadas com alteração de IF. Se não for permitido à tarefa virtual 8086 alterar IF, o monitor virtual terá que simular as instruções PUSHF, POPF, INT n, IRET, STI e CLI. Para identificar qual a instrução que terá que simular, basta examinar o conteúdo da posição de memória apontada por CS:EIP que estão na pilha do manipulador.

É potencialmente perigoso permitir que uma tarefa virtual 8086 tenha IOPL 3 pois essa tarefa poderá invocar todas as 256 interrupções através da instrução INT n. Se uma tarefa virtual 8086 desabilitar as interrupções por muito tempo, pode colocar em risco um sistema multitarefa impedindo o sistema operacional de acionar o mecanismo de preempção. Para evitar isto pode-se usar um 'watchdog' que acionará a interrupção não mascarável (NMI - Non Maskable Interrupt), cujo manipulador tomará as providências para cancelar a tarefa. Para evitar acesso, pode-se usar seletivamente o campo DPL igual a zero no "gate" da IDT.

4. Implementação

A implementação descrita neste trabalho é um protótipo de sistema operacional multitarefa para tempo real apoiado sobre o DOS, usando as características que os modernos processadores 80386 e 80486 oferecem. As características que mais nos interessam são as do modo virtual 8086, onde se pode executar programas desenvolvidos para o DOS e, usando as facilidades de paginação, suplantar a limitação de uso da memória em 640 Kb que impede o uso de toda a memória endereçada por esses processadores.

O objetivo que este trabalho visa atingir, ao desenvolver esse protótipo, é esclarecer os fundamentos da programação para o modo protegido dos processadores 80386 e 80486, do qual o modo virtual 8086 é um caso especial, desde a inicialização do mesmo até questões de gerência de processos e de memória paginada, necessários para um tal sistema.

Vários motivos tornam essa tarefa difícil e, entre outros, destacamos os seguintes:

- Não existe muita bibliografia disponível, no mercado nacional, para consulta sobre programação no modo protegido.
- Não existem referências que ofereçam o código fonte dos sistemas de exemplo, para consulta.
- A amplitude dos assuntos correlatos impede que se possa aprofundar nas pesquisas.

Na implementação do protótio, foram cobertos os seguintes aspectos gerais abordados neste trabalho:

- Definição de processo
- Gerência de processos
- Escalonador
- Mecanismo de troca de tarefa
- Sincronização
- Semáforos
- Mensagens
- Temporização
- Gerência de memória
- Tratamento de interrupções

Para cada um destes aspectos, foram fornecidas informações detalhadas sobre o que foi implantado com o objetivo de subsidiar futuras pesquisas nesta área. No item que trata a respeito de conclusões, ao final do trabalho, fornece indicativos dos pontos em que o sistema pode ser ampliado ou melhorado, também como forma de subsidiar pesquisas futuras.

Essa implementação também sofreu restrições durante sua definição, devido à necessidade de manter compatibilidade com a versão do sistema XDOS atualmente em uso na CEEE, conforme foi descrito anteriormente. Desta maneira, a interface dos serviços do sistema para com as aplicações teve que ser mantida na íntegra, muito embora a maneira de executar os

serviços pudesse ser livremente implementada. Alguns serviços, como o caso dos relógios, teve que ser tratado de maneira a executar de forma idêntica ao sistema atual, causando, sabidamente, perda de performance do sistema. A manutenção de compatibilidade sempre causa esse tipo de coisas, veja o próprio caso do uso da memória limitada em 640 Kb.

O código do protótipo desenvolvido se encontra no disquete anexo e as instruções para uso do mesmo se encontram no Apêndice.

4.1 Processo

Vamos tratar bastante a respeito de entidades que chamaremos de processo. O que chamamos de processo e o que constitui um processo, porém, precisa ficar bem claro para o entendimento da implementação.

Um programa executável é o resultado da compilação e ligação de um código fonte. Esse programa pode ser estruturado de duas maneiras distintas:

- para ser executado linearmente, o que é a maneira normal de execução.
- para execução concorrente, onde mais de um programa executa ao mesmo tempo (mesmo que isso seja feito de maneira artificial, como no nosso caso).

Para os fins desta implementação, processo é cada um programa executável, ou parte de um programa, no caso de um código executável com mais de um processo em sua estrutura, que execute concorrentemente com outros. Além disso, considera-se processo um ente que possua código, pilha e dados, além de um conjunto de conteúdo de registradores, de maneira que seja indivualmente completo, como se fosse um programa.

Estes processos que, individualmente, têm funções distintas, concorrem para, em seu conjunto, atingir um objetivo final. Esse resultado final desejado foi objeto de análise a qual levou a um projeto, de tal forma que sua implementação, na forma de programação concorrente ao invés de outras formas mais tradicionais, trazia mais vantagens em termos de eficiência e facilidade de implementação e manutenção.

Nessa implementação, um processo tem os seguintes componentes:

- TSS - que contém, além dos dados normais de todos os segmentos de estado de tarefas, os dados necessários para o gerenciamento das tarefas por parte do sistema.
- Descritor do TSS - alocado na GDT durante a instalação da tarefa no sistema, é usado durante a troca de tarefa.
- Descritor alias do TSS - alocado também na GDT durante a instalação da tarefa, é usado pelo sistema para atualizar e acessar as informações de gerenciamento colocadas dentro do TSS.
- Descritor da pilha de nível zero - alocado também na GDT durante a instalação da tarefa, é usado pelo sistema durante o tratamento de interrupções.
- Pilha - alocada pelo sistema DOS na área de memória 'NEAR' antes de chamar a tarefa, é usada durente a execução no modo virtual.
- Código - é o trecho executável que é apontado pelo TSS. Seu endereço é passado para o sistema durante a instalação do processo.

- Dados - que são comuns a todas as tarefas que fizerem parte de um mesmo programa executável, e cujo segmento está definido no TSS.
- Registradores - guardados no TSS e usados no endereçamento quando no modo virtual.
- Contexto de gerência - guardados no TSS, em área opcional, que são usados pelo sistema.
- Espaço de endereçamento - definido pelo registrador CR3, guardado no TSS, próprio de cada tarefa ou conjunto de tarefas, se estiverem no mesmo programa.

4.2 Escalonador

O escalonador tem por função escolher o novo processo a executar cada vez que for necessária a troca de processos. Essa troca acontece sempre que:

- termina o tempo de execução de uma tarefa.
- mesmo antes de terminar o tempo de execução da tarefa, esta bloqueia à espera de um recurso.
- durante a execução da tarefa, esta, voluntáriamente, libera o processador.

Os processos são organizados em uma estrutura auxiliar de gerenciamento chamada de Tabela de Processos. Nesta tabela estão relacionados os descritores dos processos já inicializados no sistema. Um Descritor de Processo é uma estrutura de dados que contém todas as informações relevantes para o processo de tal maneira que, ao retirar o processo de execução, se guarde aí todos os dados necessários para reativá-lo, quando for novamente sua vez de executar. De uma maneira geral, os dados que se necessita guardar são classificados em dois grupos:

- contexto de hardware que consiste basicamente do conteúdo dos registradores e
- contexto de software que consiste de informações de gerenciamento de processos como, por exemplo, prioridade, identificador, etc...

O critério empregado nesse protótipo para executar a troca de processos é o de filas de prioridade. Cada processo é colocado em uma fila de acordo com sua prioridade. Um vetor de ponteiros apontam para o primeiro processo em cada fila de prioridade. No TSS de cada processo, na área de contexto de software, encontramos um ponteiro para a próxima tarefa nesta fila de prioridade.

Na Fig. 4.1, podemos ver o arranjo de enfileiramento de processos por prioridade. Neste exemplo, os processos A, B e C estão enfileirados com prioridade zero. Note que, em cada prioridade temos uma fila circular com cada TSS de processo apontando para o TSS do seguinte na fila. Um vetor de ponteiros, correspondente cada um a uma prioridade, indica o primeiro da fila correspondente.

Um ponteiro nulo indica que não há processos na fila desta priodidade. Se, em um nível de prioridade só houver um processo, seu TSS aponta para si próprio. Ainda no exemplo, o processo D está sózinho na fila da prioridade cinco e os processos E e F estão enfileirados com prioridade nove.

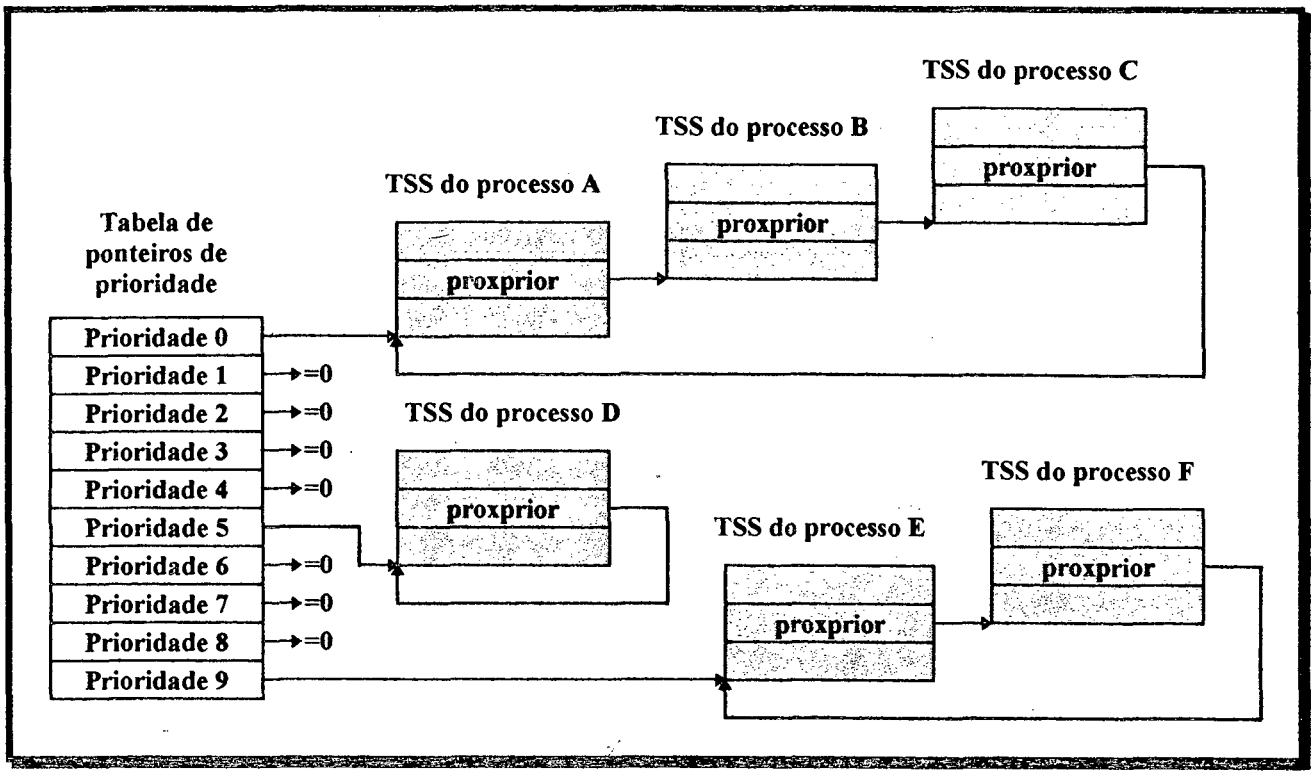

Fig. 4.1 - Disposição dos processos nas filas de prioridade

4.2.1 Tabela de Processos

Nessa implementação, a tabela de processos contém, em ordem de inicialização, os descritores dos TSS dos processos já inicializados no sistema. Nos TSS, na área opcional, disponível ao sistema operacional, colocamos todos os dados que necessitamos para gerência dos processos.

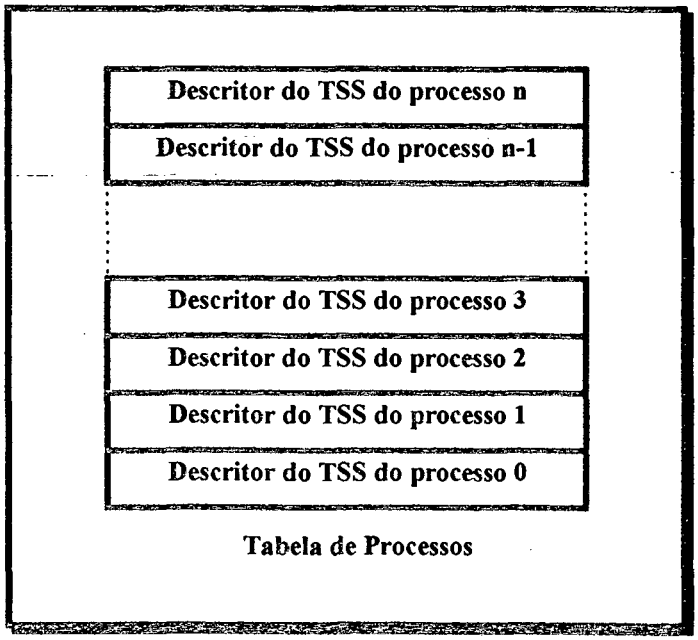

Fig. 4.2 - Tabela de Processos

4.2.2 O Segmento de Estado da Tarefa (TSS) do Processo

O Segmento de Estado da Tarefa é uma área de memória definida no projeto do processador com o objetivo de guardar o contexto do hardware durante uma troca de processo. Ao deixar de executar um processo, o processador guarda seu contexto no TSS desse processo e recupera o contexto do TSS do processo que passará a executar. Esse segmento possui um atributo específico e só serve para esta finalidade. Esse segmento não pode ser carregado em um registrador de segmento e, portanto, não pode ser lido, escrito ou executado. Uma instrução JMP ou CALL executada com um seletor que aponte para um descritor de segmento do tipo TSS causará a troca de tarefa.

O formato dos TSS, segundo a definição imposta pela arquitetura do processador, possui três áreas distintas, a primeira obrigatória e as duas últimas opcionais. A primeira área abriga o contexto do hardware, a segunda (opcional) pode conter qualquer informação que o sistema operacional necessite. A última área, apesar de opcional, também é definida pela arquitetura do processador e abriga o Mapa de Permissão de Entrada e Saida, cuja função já vimos anteriormente.

Na implementação do protótipo, o TSS dos processos, conforme pode ser visto na Fig. 4.3, usa essas três áreas distintas, cada uma com suas características e usos específicos:

- **Dados da Tarefa** usados pelo processador, cujo formato segue as regras definidas pelo projeto do mesmo e que consta de conteúdos dos registradores e outras informações relevantes do contexto do hardware, conforme detalharemos adiante.
- **Dados de Gerência do Processo** que constam de informações para gerenciamento das tarefas e de uso exclusivo do sistema operacional. Essas informações são colocadas em uma área opcional prevista pela arquitetura do processador, durante seu projeto, e que foi aproveitada com esta finalidade.
- **Mapa de Permissão de I/O** que, da mesma maneira que a primeira área, foi definida para uso do processador e, conforme seu nome indica, regula a permissão que cada processo tem de efetuar acesso a endereços de I/O.

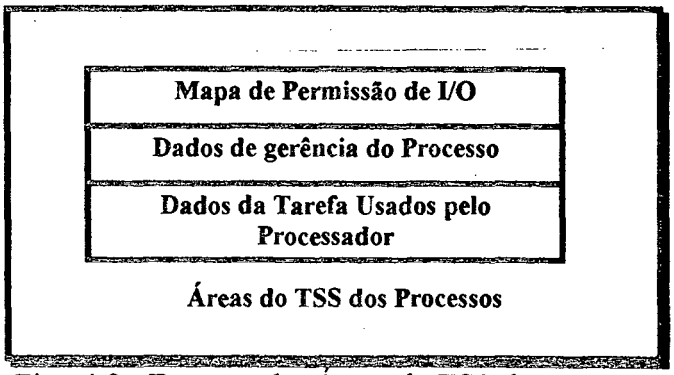

Fig. 4.3 - Formato das Áreas do TSS de Processo

Os Dados da Tarefa, que são usados pelo processador durante a troca de processos, como local para armazenar os valores relativos à situação da tarefa ativa e carregar os

valores adequados para sequência do processamento da próxima tarefa, anteriormente suspensa, são os seguintes:

- **Back-Link** usado para encadear as tarefas chamadas com intenção de retorno. Usando o conteúdo deste campo, o processador é capaz de achar a tarefa que chamou a tarefa ativa para poder retornar a ela. Essa chamada pode ter sido causada por uma instrução CALL ou pela ocorrência de uma interrupção ou exceção cujo manipulador execute com troca de tarefa ('task gate').
- **Stacks Privilegiados** (SS0, ESP0; SS1, ESP1; SS2, ESP2), um para cada nível de privilégio, usados na hipótese de uma troca de tarefa com troca de nível de privilégio. Neste caso o processador passa a usar o stack privilegiado do nível de privilégio correspondente ao da nova tarefa.
- **Registrador de Paginação** (CR3) que contém o endereço físico do diretório de páginas desta tarefa. Cada tarefa, conforme já foi visto, pode ter um diretório de páginas e as tabelas de páginas correspondentes definindo um espaço de endereçamento próprio da tarefa.
- **Ponteiro de Instrução** (EIP) indicando qual a primeira intrução que deve ser executada ao se comutar para esta tarefa. Juntamente com o valor em CS forma um ponteiro FAR para o ponto onde a execução deve ser iniciada, ou reiniciada, se esta tarefa já tiver iniciado sua execução.
- **Registrador de Flags** (EFLAGS) com os valores adequados a indicar a situação da tarefa no início, ou reinício.
- **Registradores de Uso Geral**, com os valores de eax, ebx, ecx, edx, esp, ebp, esi e edi.
- **Registradores de Segmento**, com os valores de CS, DS, SS, ES, FS e GS.
- **Descritor da LDT** da tarefa. Cada tarefa pode ter seus descritores colocados na GDT ou, opcionalmente, pode ter seus descritores colocados em uma LDT própria. No caso de ter LDT própria, o seletor da GDT que aponta para o descritor do segmento onde está a LDT deve ser colocado aqui. Caso não tenha LDT própria, deve conter zero.
- **Bit de TRAP** que permite depuração da tarefa. Se for um, cada vez que o escalonador vier para esta tarefa, o processador gerará uma exceção de debug. Tal característica é interessante para depuração dos sistemas.
- **Base do Mapa de I/O** contendo o deslocamento, em relação ao início do TSS, do início do Mapa de Permissão de I/O. Isso cria a possibilidade de, usando valores adequados, colocar quaisquer informações dentro do TSS para uso do sistema operacional sem que o processador tome conhecimento delas. Na área, criada desta maneira, colocamos os Dados de Gerência do Processo.

Os Dados de Gerência do Processo que são colocados à disposição do sistema operacional para gerenciar os processos e servir para uso das funções de interface são:

- **Identificador** - Contém o identificador interno do processo. Esse identificador é um número atribuído ao processo quando da sua inicialização e é usado como índice do processo na Tabela de Processos.
- **Estado** - Descreve a situação do processo dentre os estados possíveis de assumir de acordo com o sistema operacional. Esses estados são:

[1] pronto - quando o processo estiver apto para executar, esperando apenas sua vez na ordem de prioridade.

[2] bloqueado - quando o processo estiver esperando algum recurso do sistema que não esteja disponível. Quando o recurso for disponibilizado, o processo passará ao estado pronto.

[3] bloqueado esperando mensagem - quando o processo estiver esperando a chegada de uma mensagem. A espera pode ser de duas modalidades: vir de um processo qualquer , ou vir de um processo específico.

- **Prioridade** - Indica a prioridade deste processo para uso do escalonador ao escolher o próximo processo a ser executado.
- **Relógio** - Esse campo contém um contador de intervalos de tempo (em unidades de 55 ms) que serve como temporizador para este processo. O processo bloqueia ao ser ativado o temporizador. Esse campo leva um valor que a cada 55 ms é decrementado e, ao chegar a zero, o sistema desbloqueia o processo.
- **Semáforo** - Esse campo é usado no enfileiramento dos processos na fila de um semáforo. Ver item 4.3.1.
- **Mensagem** - Contém o endereço (segmento) da primeira mensagem na Fila de Mensagens endereçadas a este processo. Ver item 4.3.2.
- **Nova Mensagem** - Indica se chegou uma mensagem nova desde que o processo ficou bloqueado esperando mensagem de um processo específico ou se ocorreu um estouro de tempo. Ver item 4.3.2.
- **Próximo na Prioridade** - Indica o próximo processo com a mesma prioridade na fila dos processos com a prioridade deste. Ver Fig. 4.1 no início deste ítem.
- **PSP** - Contém o endereço (segmento) do PSP do programa ao qual pertence o processo.
- **Control_C** - Contém o endereço do manipulador de CTRL_C instalado pelo programa ao qual pertence o processo
- **Erro Crítico** - Contém o endereço do manipulador de erro crítico instalado pelo programa ao qual pertence o processo
- **Nome** - Contém um conjunto de quatro caracteres que serve como nome do processo com propósito de uso externo.

O Mapa de Permissão de I/O é definido como um conjunto de bits correspondendo cada um a um endereço de I/O. O valor um no bit indica acesso permitido ao endereço e o valor zero no bit indica acesso negado. Se uma tarefa tentar acessar um endereço de I/O que não lhe é permitido, o processador gerará uma exceção.

O conteúdo do TSS de um processo pode ser visto na Fig. 4.4.

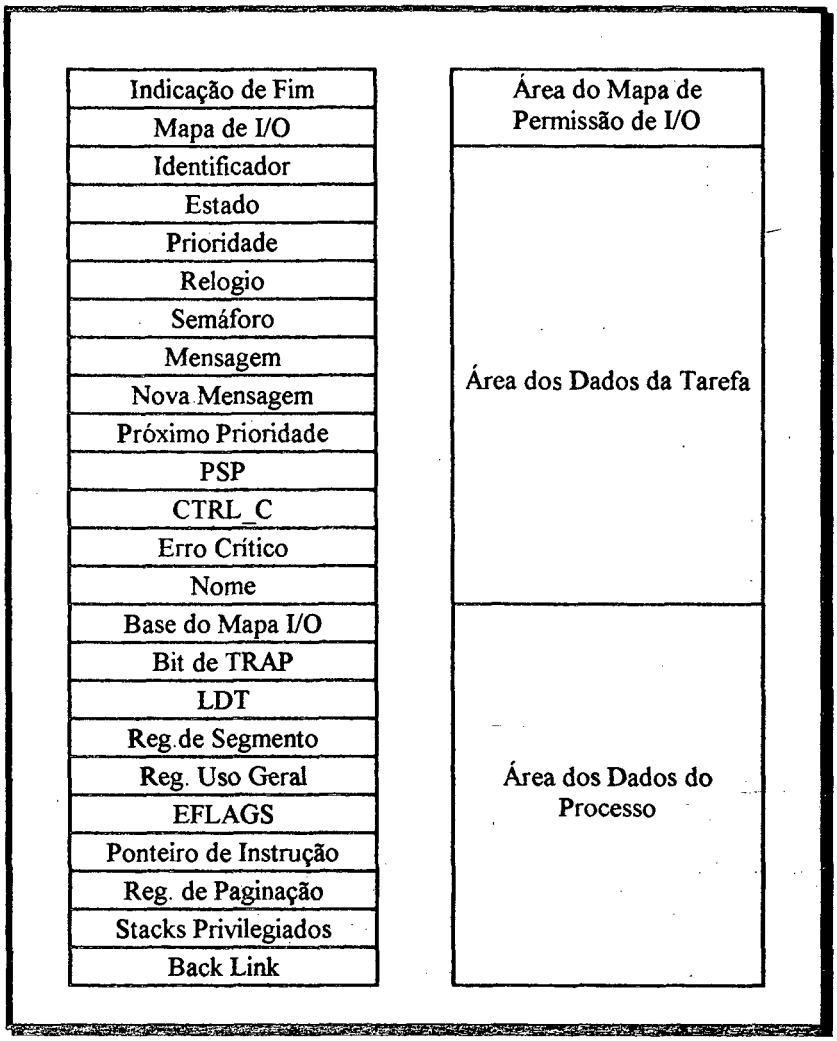

Fig. 4.4 - Dados colocados em cada área do TSS dos processos

4.2.3 Mecanismo de Troca de Tarefa

A troca de tarefas é gerenciada pelo escalonador, que é chamado pelo Monitor Virtual através de uma instrução CALL. Conforme vimos anteriormente, essa troca acontece sempre que:

- termina o tempo de execução de uma tarefa.
- mesmo antes de terminar o tempo de execução da tarefa, esta bloqueia à espera de um recurso.
- durante a execução da tarefa, esta, voluntariamente, libera o processador.

O controle do término de tempo de execução é feito pelo Monitor Virtual utilizando uma variável contadora de interrupções do relógio do computador. Portanto nosso escalonador está vinculado ao relógio do computador.

Cada vez que é chamada a interrupção do relógio, o Monitor Virtual providencia o retorno ao modo virtual para tratamento da interrupção pela rotina original instalada no vetor de interrupção do DOS e o retorno ao modo protegido para seqüência das operações de tratamento dos relógios do sistema. Depois, decrementa o contador para verificar se está no final

do tempo determinado para a tarefa ativa executar e, se estiver, chama o escalonador. Este procura um processo com prioridade maior que o processo que estava executando e, se encontrar, passa a execução para ele.

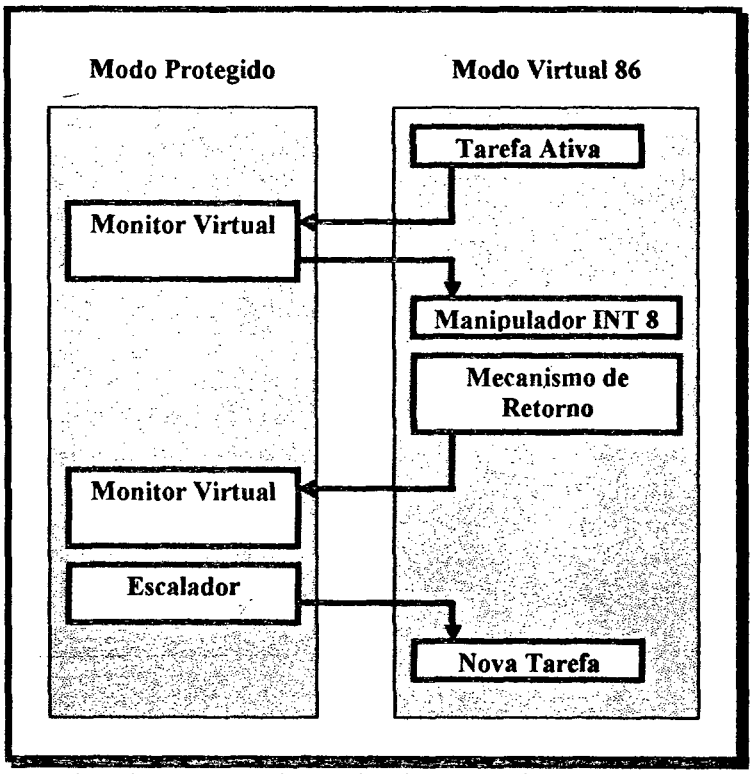

Fig. 4.5 - Mecanismo de chamada do escalonador

Cada uma destas etapas tem suas características e vamos passar a vê-las em detalhes. Iniciando com a chamada da INT 8, interrupção do relógio. Ao ser interrompido, o processador automaticamente troca de modo de operação saindo do modo virtual 86 e indo para o modo protegido. No modo protegido, o processador busca, através do registrador IDTR e da IDT o manipulador para a interrupção ocorrida. Neste processo, através de mecanismos já vistos anteriormente quando tratamos de interrupções, a execução chega ao Monitor Virtual.

Devido à troca de nível de privilégio ao passar do modo virtual 86 (nível de privilégio 3) para o modo protegido (nível de privilégio 0), o processador coloca no stack de nível zero da tarefa interrompida (cada tarefa tem seu próprio stack de nível zero) o conteúdo de vários dos registradores. Na Fig. 4.6 pode-se ver a situação da pilha de nível zero, neste instante.

Adicionalmente, o Monitor Virtual guarda, na mesma pilha, o conteúdo dos registradores de uso geral para poder dispor destes registradores para seu próprio uso. O perfil do stack gerado até este momento é o mostrado na Fig. 4.6, ressaltando que os valores desde GS até EIP são colocados pelo processador ao trocar de nível de privilégio e os demais são colocados pelo Monitor Virtual.

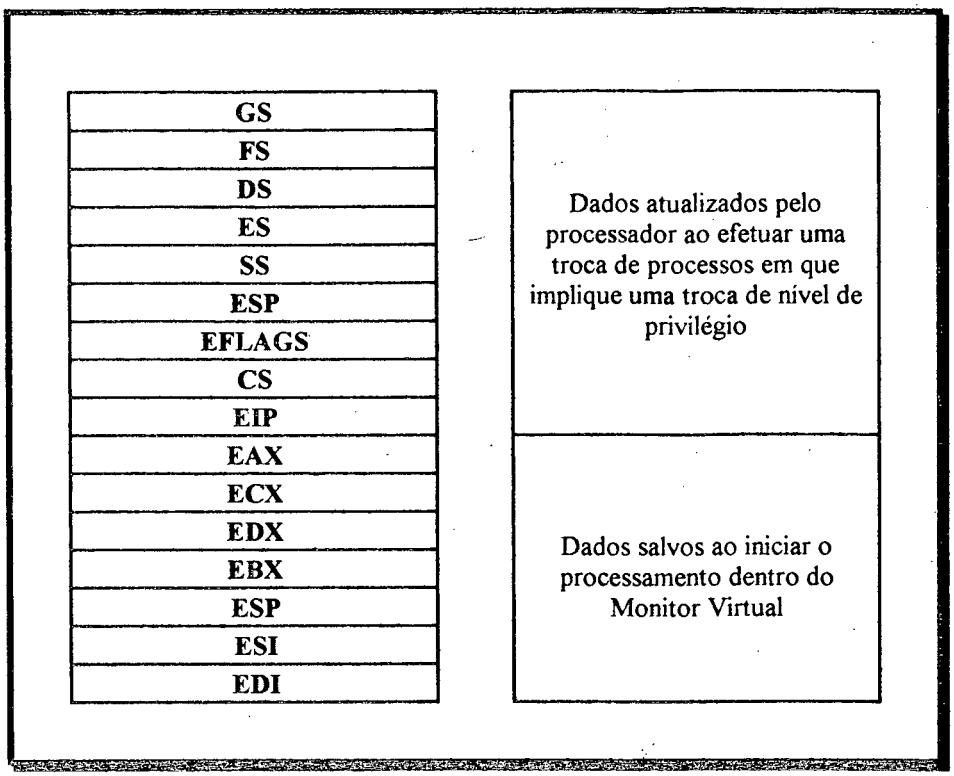

Fig. 4.6 - Situação do stack de nível zero no início do monitor virtual ao sofrer uma INT 8.

Identificada como sendo a INT8 a causadora da interrupção, o procedimento é voltar ao modo virtual 86 para tratá-la através do manipulador instalado no sistema de tratamento de interrupções do DOS. O procedimento normal, visto no tratamento de interrupções, é colocar o endereço do manipulador no stack de nível protegido, no lugar do CS:EIP lá existente. Este endereço, indicando a posição da interrupção da tarefa ativa, é transferido para o stack de nível 3 do processo, em posições adicionadas ao mesmo. Isso feito, podemos executar um retorno de interrupção para voltar ao modo virtual 86, executar o manipulador da INT 8 do DOS e voltar a executar a tarefa interrompida.

Porém não é isso que desejamos. Nós queremos voltar ao modo protegido após executar o manipulador da INT 8 do DOS para tratar dos relógios do sistema e para executar a troca de tarefas, se estiver na hora. Para isso não colocamos no stack de nível 3 da tarefa interrompida o endereço de retorno à ela mas sim o endereço de um trecho de código que chama uma interrupção conhecida que chamamos de Interrupção de Retorno ao Modo Protegido e salvamos o endereço de retorno à tarefa interrompida em uma variável para uso posterior. Ao se executado este trecho de código, voltamos ao modo protegido e ao Monitor Virtual. Este, reconhecendo a interrupção especial, recupera o endereço guardado recolocando-o no stack e passa a tratar os relógios do sistema.

Nesse ponto, o conteúdo do stack de nível zero continua igual ao visto acima. Todas as manipulações executadas, o retorno ao modo virtual e a volta ao modo protegido foram acompanhados de retiradas de informações do stack de nível zero e do stack do modo virtual, mas a situação neste instante continua a mesma, com valores diferentes, possivelmente, devido ao atendimento da INT8, mas formalmente a mesma situação.

Depois de tratar dos relógios do sistema,da maneira que veremos adiante, o Monitor Virtual verifica se está na hora de trocar de processo. Se for hora, chama o escalonador através de uma instrução CALL, o que acrecenta o endereço de retorno ao stack de nível zero.

O escalonador executa uma busca nas filas de prioridade, iniciando pela fila de prioridade mais alta, até achar um processo em condições de ser executado. Definido qual o processo a ser executado, o escalonador verifica se este processo escolhido não é o mesmo que está ativo. Se for, executa um retorno ao Monitor Virtual que executa um retorno da interrupção de tratamento da INT 8, que estava sendo tratada, voltando ao modo virtual no ponto de interrupção. Se não for, executa um salto para o TSS da nova tarefa, o que salvará o contexto da tarefa atual em seu TSS.

Esse salto para o TSS da nova tarefa, em sua primeira vez, fará com que a execução inicie, no modo virtual, em uma situação de acordo com os valores colocados no TSS da tarefa, por ocasião de sua inicialização. Nas demais vezes, a execução será reiniciada no modo protegido, dentro do escalonador, no comando imediatamente posterior ao salto para o TSS da nova tarefa. Este comando é um IRET que levará a execução de volta ao Monitor Virtual que fará, por sua vez, um RET que levará ao modo virtual no ponto em que o processo foi interrompido por uma INT8 e perdeu o processador.

Para uma tarefa interrompida pela INT8 e que perdeu o processador, tudo se passa como se o manipulador desta interrupção tivesse levado um tempo enorme e indeterminado para atendê-la. Nesse tempo indeterminado, possivelmente vários outros processos de prioridade mais alta foram executados e várias outras interrupções, INT8 inclusive, foram atendidas.

De acordo com tudo o que foi visto acima, no instante em que a tarefa é escalada, já tendo sido interrompida anteriormente, o conteúdo do stack de nível zero é conforme a figura abaixo.

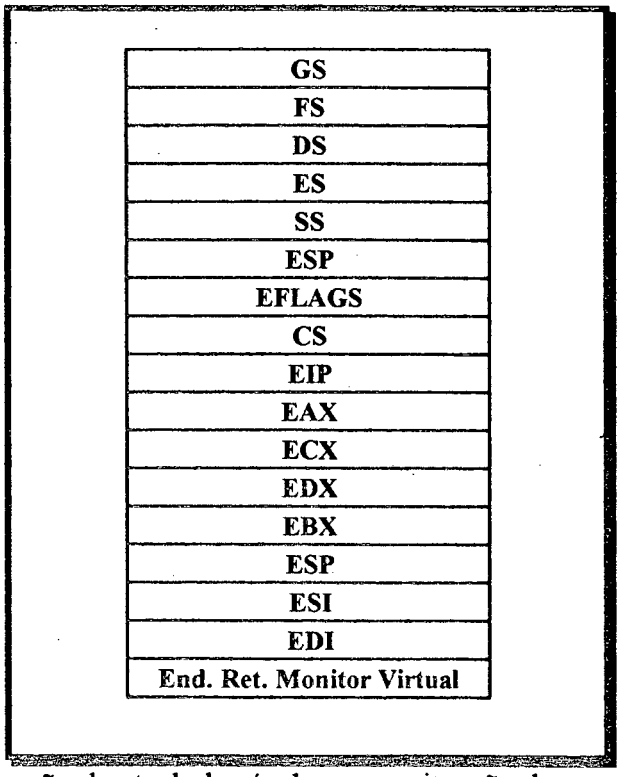

Fig. 4.7 - Situação do stack de nível zero na situação de reescalonamento.

4.3 Sincronização

Em programação concorrente sempre surge a necessidade de sincronizar dois processos. Os mecanismos mais comuns que se pode empregar para isso são os semáforos e a troca de mensagens. No sistema descrito neste trabalho, implementou-se esses dois tipos de mecanismos de sincronização.

4.3.1 Semáforos

Em nossa implementação os semáforos são do tipo semáforos contadores. Sua estrutura é constituida de:

- **Contador** - que recebe, durante a inicialização, o número de recursos disponíveis dentro desta área crítica, ou seja, o número de processos que podem entrar nesta área controlada.
- **Fila** - correspondendo ao seletor do TSS primeiro processo da fila dos processos bloqueados nesse semáforo, esperando pela liberação do trecho crítico ou do recurso. Se for igual a zero, não existem processos bloqueados na fila deste semáforo.

A fila de processos bloqueados em um semáforo fica organizada com o primeiro processo bloqueado sendo indicado pelo campo FILA do semáforo. No descritor do primeiro processo bloqueado, no campo SEMAFORO encontramos o seletor do TSS do processo seguinte da fila e assim por diante até encontrarmos um processo com o valor zero neste campo, o que indicará ser este processo o último da fila. No caso de não haver processo bloqueado no semáforo, o conteudo do campo FILA é igual a zero. Na Fig. 4.8, mostramos um semáforo, em cuja fila estão bloqueados os processos x e y, nesta ordem.

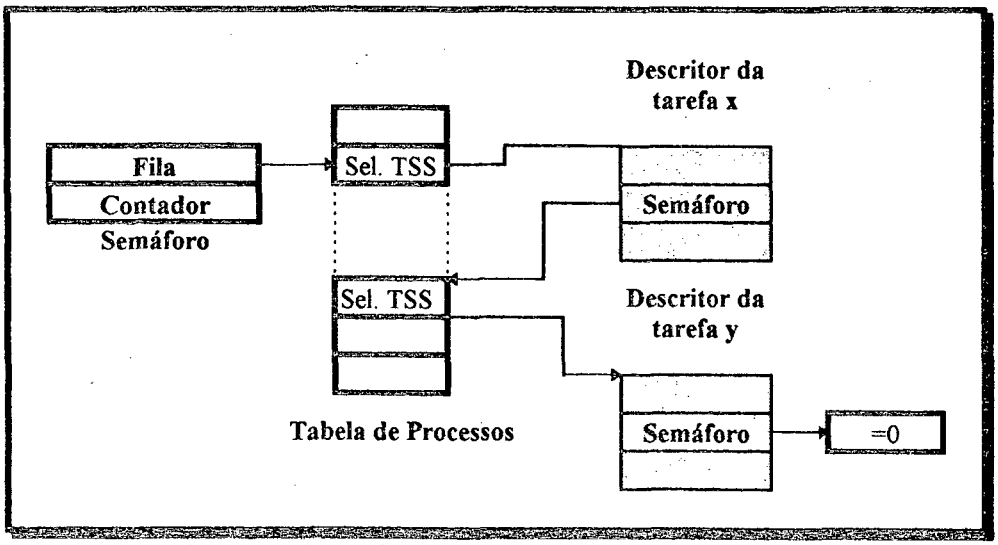

Fig. 4.8 - Semáforos e o encadeamento de sua fila.

As operações definidas para os semáforos são:

- **Inicialização** - A operação de inicialização cria uma variável semáforo, inicializa o campo do contador com um valor fornecido pelo aplicativo e coloca zero no campo FILA, indicando não haver processos bloqueados.

- **P** - A operação P decrementa o valor do contador e testa seu valor. Se for positivo ou zero, permite que o processo chamador prossiga. Se o valor do contador for negativo, bloqueia o processo que a chamou, liberando o processador para o seguinte, na ordem da prioridade.
- **V** - A operação V incrementa o contador e testa seu valor. Se o valor do contador for negativo, significa que existem processos esperando na fila deste semáforo. Se houverem processos esperando na fila do semáforo, o primeiro deles será liberado.

4.3.2 Mensagens

Mensagens são tratadas como áreas de memória de tamanho variável, alocadas pelo sistema, cujo endereço é entregue ao processo receptor. Quando um processo chama os serviços de transferência de mensagens, o sistema aloca uma área do gerenciador de memória e copia a mensagem da área de memória do processo transmissor para esta área de memória do sistema. Após copiar a mensagem para a área alocada, o sistema coloca o endereço desta área na fila de mensagens do processo destino.

Quando a execução retornar ao aplicativo que enviou a mensagem, este já poderá dispor da sua área de memória usada para armazenar a mensagem, pois a mensagem já foi copiada para uma área de memória do sistema. A tarefa que vai receber a mensagem, recebe o endereço da área alocada do sistema, devendo liberá-la após uso.

O formato da mensagem, no que interessa ao aplicativo que a está enviando, é livre. Enquanto está a cargo do sistema, porém, a mensagem recebe dois campos adicionais, de maneira que fica com os seguintes campos:

- **Tamanho** - neste campo, com dois bytes, temos o tamanho, também em bytes, da mensagem original copiada pelo sistema da área do processo transmissor.
- **Mensagem** - este campo contém a mensagem que está sendo transmitida, no formato particular que deve ser acertado entre o transmissor e o receptor. O sistema não toma conhecimento deste protocolo e apenas copia a mensagem, byte a byte da área do transmissor para uma área do sistema.
- **Endereço** - este campo contém o endereço da próxima mensagem dentro do encadeamento das mensagens recebidas pelo processo receptor. No caso desta mensagem ser a última, este campo contém zero. O processo receptor, após tratar a mensagem, deve liberar esta área para o sistema, através de uma função da interface com os aplicativos.

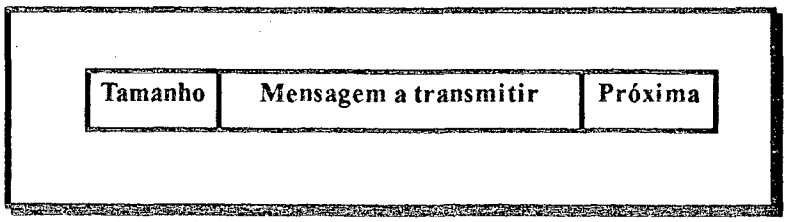

Fig. 4.9 - Formato da Mensagem

Com relação ao processo receptor, o sistema coloca a mensagem na fila de mensagens recebida pelo mesmo. O endereço da primeira mensagem nesse encadeamento está no TSS do processo receptor. No campo MENSAGEM está um endereço que aponta para a primeira mensagem na fila das mensagens recebidas pelo processo. Se este valor for zero, não

existe mensagem recebida e, portanto, não tem fila. As demais mensagens da fila são encontradas através do campo de encadeamento que cada mensagem possui. Na Fig. 4.10, vemos o processo x com tres mensagens encadeadas.

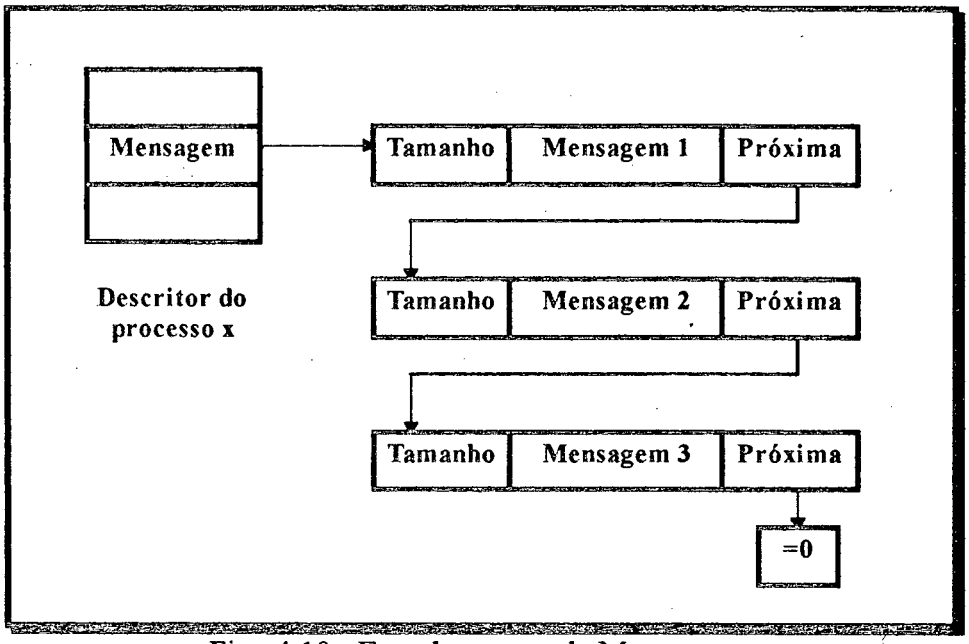

Fig. 4.10 - Encadeamento de Mensagens.

As operações definidas para a troca de mensagens entre processos, na interface do sistema, são:

- **Send** - Envia uma mensagem para um processo. A mensagem é copiada para uma área alocada do sistema cujo endereço é encadeado na fila de processos recebidos pelo processo receptor.
- **WaitT** - Retira a primeira mensagem da fila de mensagens do processo. Se não houver mensagens na fila, bloqueia o processo durante um tempo definido. O processo pode ser desbloqueado por estouro de tempo ou pela recepção de uma mensagem.
- **Sleep** - Espera uma mensagem de um processo específico, durante um tempo determinado. Se, durante este tempo, não chegar mensagem deste processo, retorna por estouro de tempo.
- **Wake** - Envia mensagem para um processo, colocando-a no início da fila e indicando, através de um campo específico no TSS do processo, que a mensagem colocada no início da fila, foi enviada por esta modalidade.

O conjunto de funções Send e WaitT funcionam como transferência de mensagens sem sincronização, enquanto o conjunto Sleep e Wake causam a sincronização dos dois processos. Essas funções devem ser utilizadas aos pares, conforme pode ser visto na Fig. 4.11.

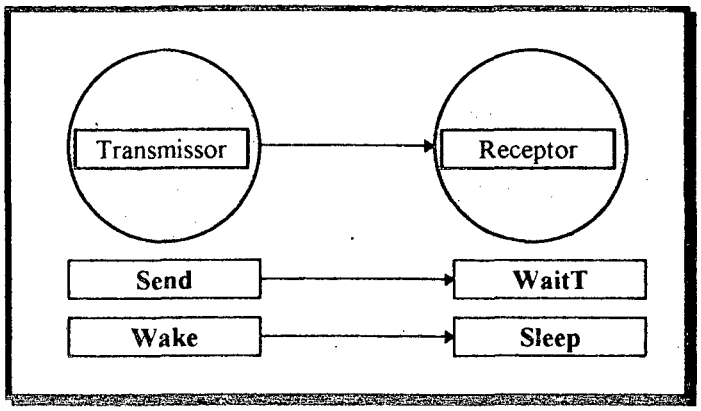

Fig. 4.11 - Funções de transferência de mensagens

4.4 Temporização

Várias situações dentro da programação concorrente nos obriga a lançar mão de temporizações. O caso mais comum são os protocolos de comunicação onde um processo encarregado da transmissão da mensagem, após enviá-la, fica bloqueado à espera da mensagem indicativa da recepção, por parte do processo receptor, da mensagem enviada. Essa espera deve ser temporizada, para que, se não for recebida a mensagem de reconhecimento em um tempo definido, o processo transmissor tome a decisão de re-enviá-la, se isso for adequado. Essa temporização foi implementada através de relógios.

4.4.1 Relógios

Os relógios são estruturas de dados fornecidas e gerenciadas pelo sistema, sendo constituidas pelos seguintes campos:

- **Contador** - Contém o número de intervalos de 55 ms que ainda restam até que o relógio atue causando a execução da rotina associada.
- **Rotina Segmento** - Contém o segmento do endereço da rotina associada.
- **Rotina Offset** - Contém o deslocamento do endereço da rotina associada.
- **Parametro** - Contém um valor que será passado à rotina de interrupção por ocasião da atuação do relógio.

As funções fornecidas, pela interface do sistema, que atuam sobre os relógios são as seguintes:

- **Cria** - Busca um relógio disponível no conjunto de relógios do sistema e inicializa a estrutura dele com as informações fornecidas.
- **Parte** - Inicializa o contador, ativando o relógio.
- **Congela** - Pára a contagem do tempo, mantendo o valor atual do contador.
- **Descongela** - Reativa a contagem do tempo.
- **Para** - Suspende a contagem do tempo, desativando o relógio.
- **Libera** - Devolve o relógio ao gerenciador para poder ser utilizado por outros processos.

O tratamento dos relógios é feito, então, da seguinte maneira: o relógio é criado e é dada a partida ao mesmo. O sistema, dentro do Monitor Virtual, a cada ocorrência da INT8, depois de voltar ao modo virtual para executar a rotina de tratamento da BIOS e retornar ao modo protegido, decrementa o contador dos relógios ativos e testa o valor resultante. Se o valor chegou em zero, é hora de atuar. Nesse caso, o sistema retorna ao modo virtual de forma semelhante à usada para tratamento da INT 8 com a diferença que a INT chamada para retornar ao modo protegido é outra, caracterizando retorno após executar rotina de relógio. O Monitor Virtual, reconhecendo esta interrupção especial, salta para o código de tratamento de relógios para tratar os restantes. A Fig. 4.12 ilustra este tratamento.

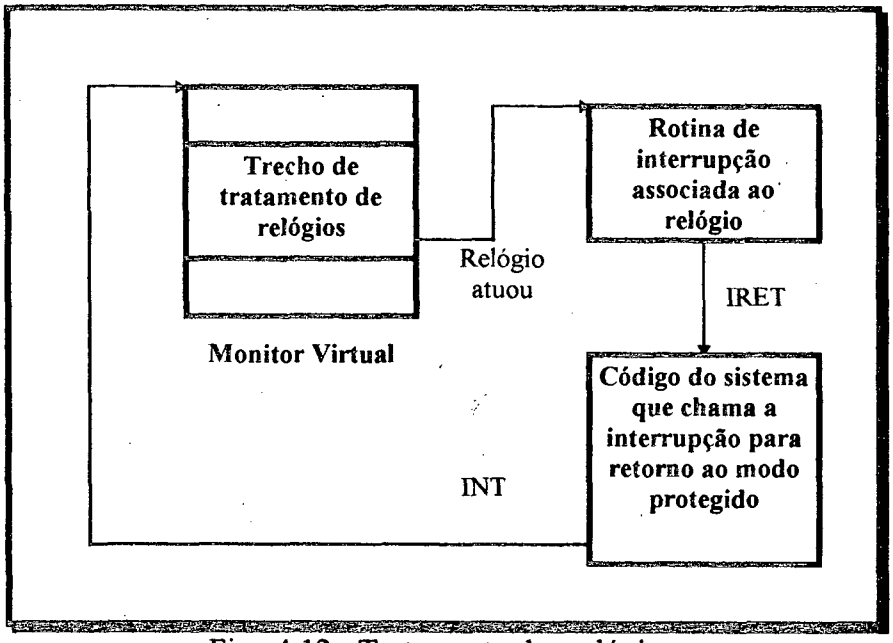

Fig. 4.12 - Tratamento dos relógios.

4.5 Memória

O gerenciamento de memória se desdobra em três partes distintas:

- **Área de memória baixa**, alocada durante a inicialização do sistema através de uma chamada ao DOS, se destina a alocações para as pilhas usadas pelos processos durante a execução no modo virtual e para as mensagens transferidas entre os processos.
- **Área de memória alta**, reservada acima dos endereços acessáveis pelo DOS, se destina a pilhas usadas pelos processos durante o tratamento de interrupções no modo protegido e para os TSS dos processos.
- **Área de memória de paginação**, restante da memória acima da memória alta e que fica destinada a paginação, expandindo o trecho de memória usado pelo DOS para alocar código executável e para alocação das estruturas necessárias á paginação

O limite superior da área destinada aos programas aplicativos é fixa e foi definida pelo projeto do DOS e o limite inferior depende dos aplicativos residentes antes da inicialização do sistema de paginação.

O limite inferior da área de memória alta foi escolhido de maneira a não impedir a utilização da chamada Área de Memória Alta do DOS(HMA - High Memory Area) e o limite superior fica definido pelo tamanho desta área.

O limite inferior da área de memória de paginação fica definido pela posição da área de memória alta e seu limite superior depende da quantidade de memória instalada no computador. A Fig. 4.13 ilustra essas áreas de memória e seus limites.

As áreas de memória baixa e de memória alta compartilham o mesmo gerenciador porque sua organização é semelhante, diferindo apenas no descritor de segmento da área de memória. A área de memória de paginação, por suas características diversas, possui seu próprio gerenciador.

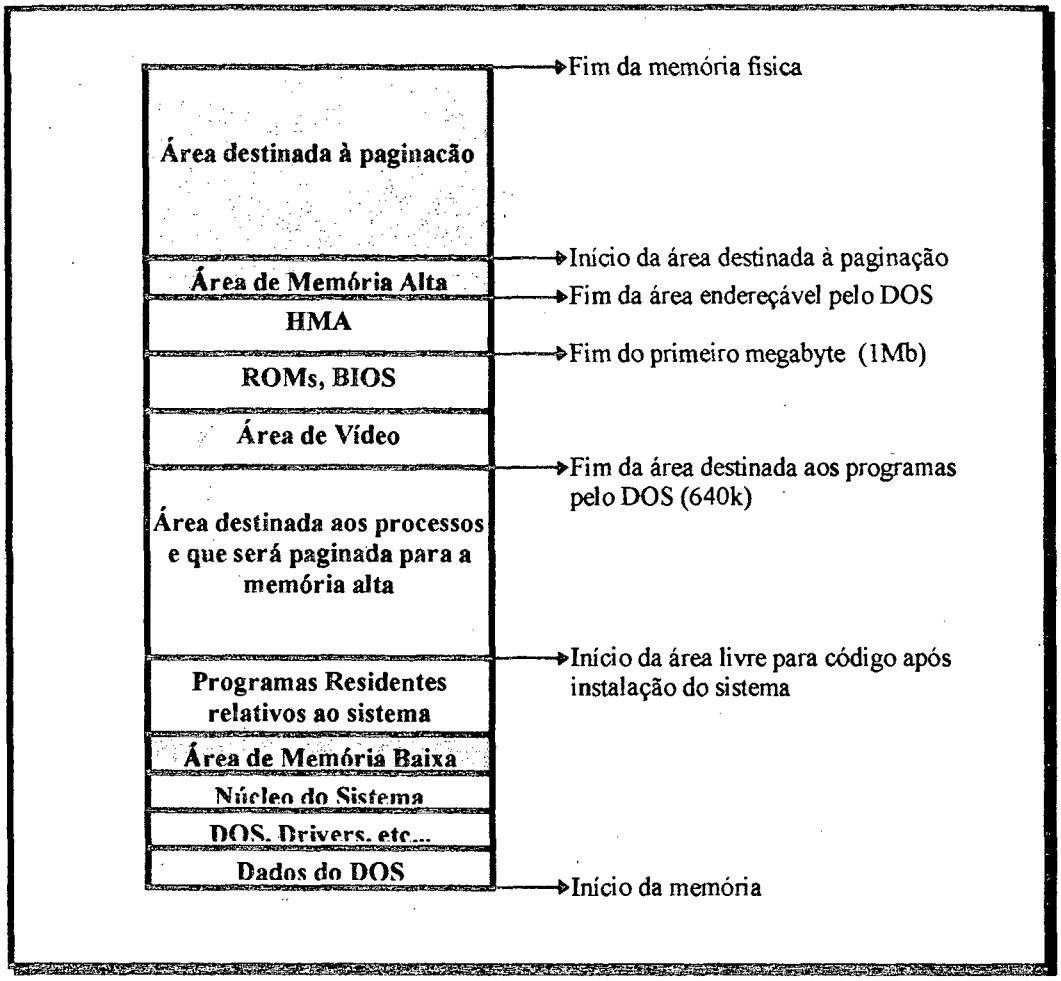

Fig. 4.13 - Disposição das áreas de memória.

4.5.1 Gerência das Áreas de Memória Baixa e de Memória Alta

A gerência dessas duas áreas de memória está baseada em um cabeçalho para cada bloco alocado. Esse cabeçalho, com tamanho de um parágrafo, contém as seguintes informações:

- **Gabarito** - contém uma palavra com o valor hexadecimal 55AA, indicando um cabeçalho válido de bloco de memória.
- **Estado** - indica se o bloco está livre ou ocupado.
- **Tamanho** - informa o tamanho do bloco, em parágrafos, sem levar em consideração o cabeçalho.

- **Próximo** - ponteiro para o início do bloco seguinte. Aponta para o bloco alocado e não para o cabeçalho do bloco. Esse é um endereço de segmento, alinhado em parágrafo.
- **Anterior** - ponteiro para o início do bloco anterior. Aponta para o bloco alocado e não para o cabeçalho do bloco. Esse é um endereço de segmento, alinhado em parágrafo.
- **Disponível** - espaço não utilizado no cabeçalho.

Na Fig. 4.14 mostramos o conteúdo do cabeçalho e a forma de encadeamento dos blocos.

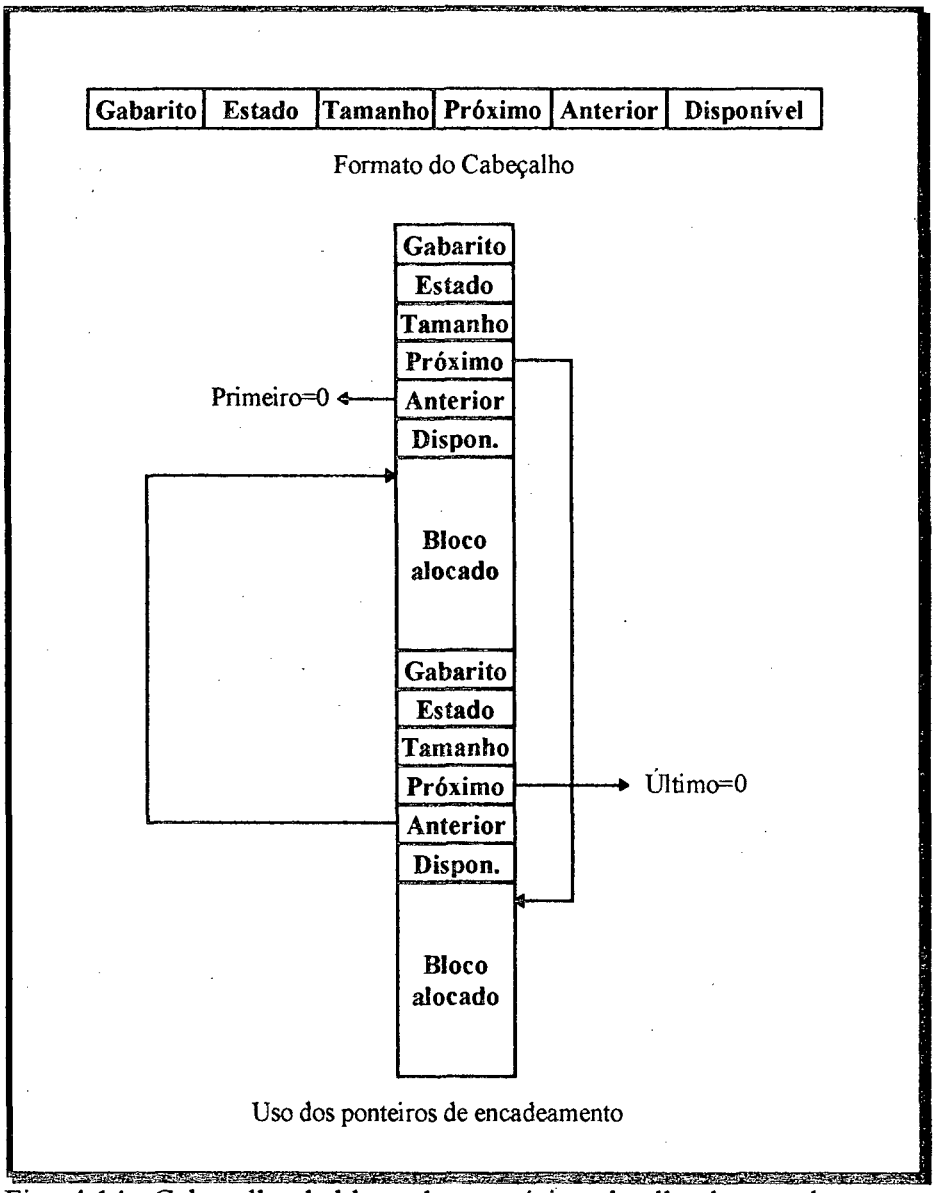

Fig. 4.14 - Cabeçalho de bloco de memória e detalhe do encadeamento

O gerenciamento desta área é feito através do uso de quatro funções básicas, internas e que não estão disponíveis na interface do sistema. Essas funções auxiliares são:

- **Aloca memória baixa** - Essa rotina retorna em AX o endereço de segmento (offset = 0) do bloco alocado dentro da área de memória baixa. Deve receber o tamanho do bloco solicitado.

- **Aloca memória alta** - Essa rotina retorna um offset de 32 bits para o bloco alocado dentro do segmento reservado para uso do sistema na área de memória alta. Deve receber o tamanho do bloco solicitado.
- **Desaloca memória baixa** - Essa rotina libera um bloco de memória alocado na área de memória baixa cujo endereço é fornecido. Retorna um código indicando sucesso ou falha na desalocação. Após desalocar o bloco, a rotina tenta aglutiná-lo aos blocos adjacentes livres, se houverem, evitando a fragmentação da memória.
- **Desaloca memória alta** - Essa rotina libera um bloco de memória alocado na área de memória alta cujo seletor na GDT é fornecido. Retorna um código indicando sucesso ou falha na desalocação. Após desalocar o bloco, a rotina tenta aglutiná-lo aos blocos adjacentes livres, se houverem, evitando a fragmentação da memória.

As rotinas de interface que tratam do gerenciamento de memória se restringem à área de memória baixa, única de interesse dos aplicativos e são apenas duas:

- **Aloca** - que serve para solicitar uma área de memória ao gerenciador de memória baixa.
- **Desaloca** - que serve para devolver, ao gerenciador de memória baixa, uma área de memória anteriormente alocada.

4.5.2 Gerência da Alocação da Área de Memória de Paginação

Durante a inicialização do sistema, toda a memória existente no computador é mapeada através de um diretório de páginas e de tantas tabelas de páginas quantas necessárias. Essas estruturas mapeiam a memória de tal forma que a tradução dos endereços lineares gerados pelos descritores de segmentos no modo protegido ou pelos segmentos e offsets no modo virtual são traduzidos pelo mecanismos de paginação com uma relação de um para um, ou seja, os endereços lineares são idênticos aos endereços físicos. Após essas estruturas serem inicializadas, o registrador CR3 é carregado com o endereço físico do diretório de páginas e a paginação é habilitada através do registrador CR0.

Algumas áreas de memória já estão ocupadas e tem sua posição definida. Essas áreas são: área de dados do DOS, área dos programas residentes e device drivers, área ocupada pelo núcleo do sistema, enfim, tudo o que está na memória até que os mecansismos de paginação sejam acionados. Acima destas áreas está a área destinada aos programas aplicativos que irão se beneficiar da paginação e compartilhar o DOS, o sistema, o núcleo multitarefa e os demais drivers de dispositivo já instalados.

Esta área é definida e mapeada até o limite de 640 kb. Acima desta área, temos de novo posições de memória que estão ocupadas pelo DOS, BIOS ou outros componentes do sistema, até o limite da área conhecida como HMA (1 Mb + 64Kb - 16 b), que também serão compartilhadas.

Acima desta área temos área de memória alta para alocação do sistema e acima desta, a área de memória de paginação, que é preparada para uso. A Fig. 4.15 mostra a disposição destas áreas com maiores detalhes.

O mecanismo de troca de ambiente de endereçamento é acionado através da troca do conteudo do registrador CR3, o que é feito automáticamente cada vez que se troca de tarefa executando um JMP para um TSS. O valor do registrador CR3 é carregado do campo correspondente do TSS.

Esse mecanismo, entretanto, não é acionado durante a ocorrência de uma interrupção, porque nosso sistema usa 'interrupt gates' para as rotinas de tratamento e não 'task gates', quando ocorreria troca de tarefa para tratá-las. Desta forma, as rotinas de tratamento de interrupção, instaladas no modo virtual, devem estar em posições de memória que sejam comuns a todos os ambientes de endereçamento de todas as tarefas, pois uma interrupção pode ocorrer em qualquer uma delas e seu tratamento ocorrerá no ambiente da tarefa interrompida e, portanto, no seu espaço de endereçamento.

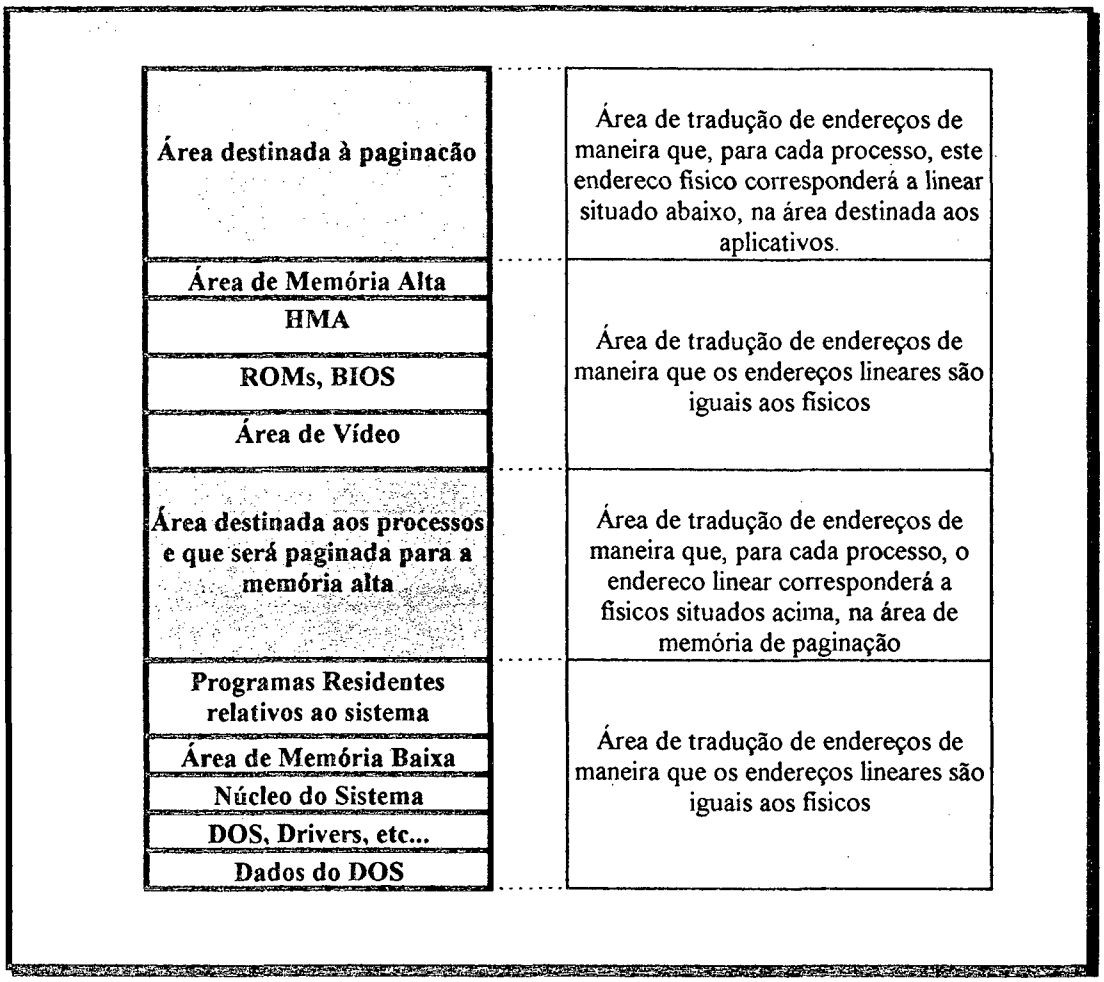

Fig. 4.15 - Disposição das áreas de memória afetadas pelo mecanismos de paginação.

Devido ao exposto, todas as rotinas de tratamento de interrupção de sejam instaladas pelo DOS ou pelo ambiente operacional, ou ainda por tarefas, deverão estar posicionadas na memória antes da inicialização do sistema de gerenciamento da paginação.

A inicialização do gerenciamento da paginação consiste em determinar, na área destinada aos aplicativos, o início e o número de páginas a serem gerenciadas e, na área destinada à memória de paginação, na memória alta, o número de páginas disponíveis para paginação.

Na área destinada aos aplicativos, basta procurar, usando a cadeia de alocação de memória do DOS (MCB), o último bloco, que será um bloco livre, de grande tamanho, situado adiante, após o MCB correspondente ao PSP do núcleo do sistema. Esse endereço, que será o endereço inicial que procuramos, não estará situado em uma fronteira de página, porém, se tomarmos o cuidado de sempre copiar a primeira página ao trocar o ambiente de endereçamento

para que tenhamos esse trecho compartilhado por todos eles, podemos usar a fronteira de páginas imediatamente anterior. O endereço final é fixado pelo DOS e está situado nos 640 kb. O número de páginas fica definido pelo cálculo. A Fig. 4.16 ilustra essa situação.

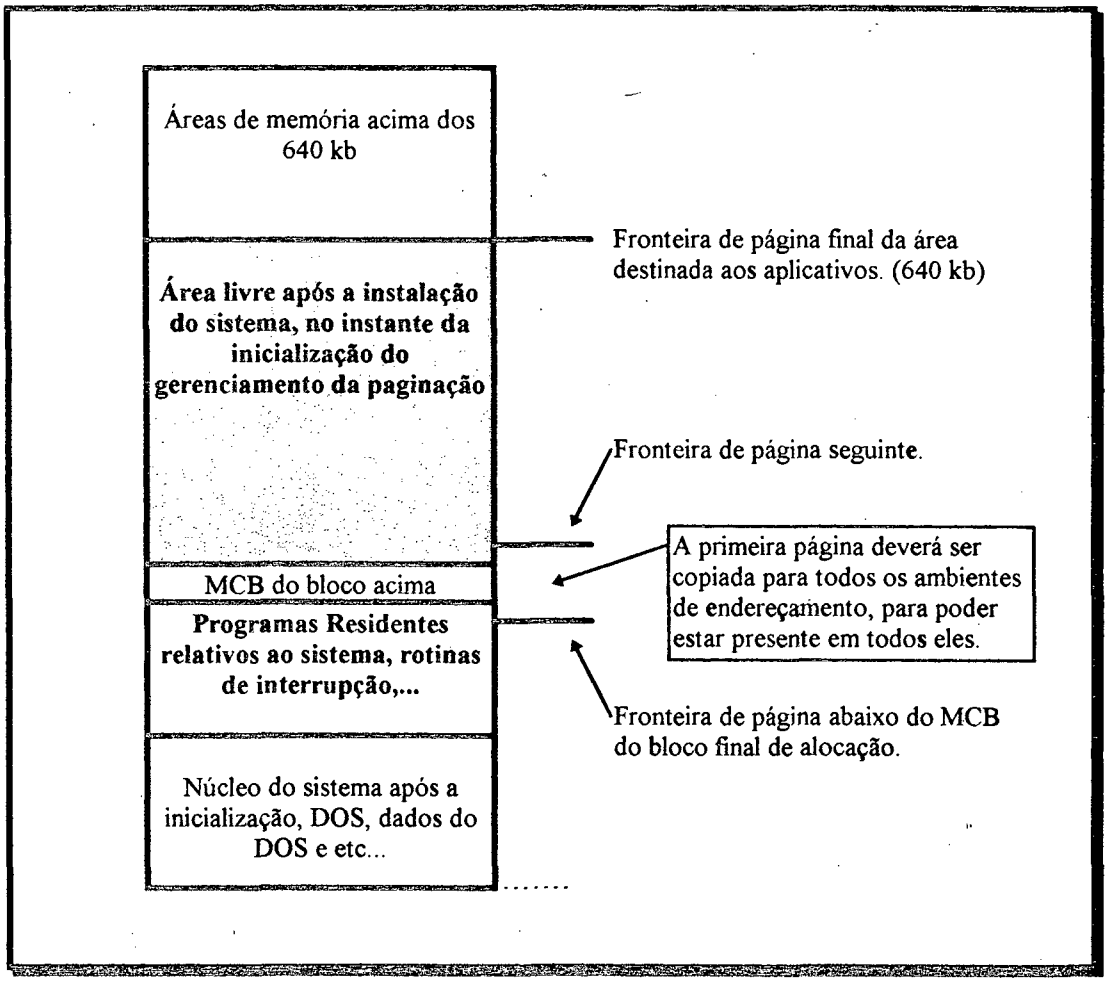

Fig. 4.16 - Detalhes da inicialização do gerenciador de paginação.

Na área de memória de paginação, a inicialização é mais fácil. Durante a inicialização do sistema, ainda no modo real, uma chamada ao BIOS (INT 15h, função 88h) determina o tamanho da memória estendida, definindo o limite superior da memória. O limite inferior é definido pelo próprio sistema e está situado após a área destinada ao gerenciador da memória alta no endereço hexadecimal 120000.

O controle do uso desta área é feito através de um conjunto de bits onde cada um corresponde a uma página e, se este for igual a um, indica página ocupada e se for zero indica página disponível. A Fig. 4.17 ilustra a posição desta área de memória.

Depois disso, copia o diretório de páginas e as tabelas de páginas para páginas livres da memória de paginação, alocadas do sistema de gerenciamento da paginação através de rotinas auxiliares, e as corrige para que as páginas ocupadas pelo aplicativo que chamou a função sejam remapeadas para novas páginas livres da memória de paginação também alocadas do sistema de gerenciamento da paginação.

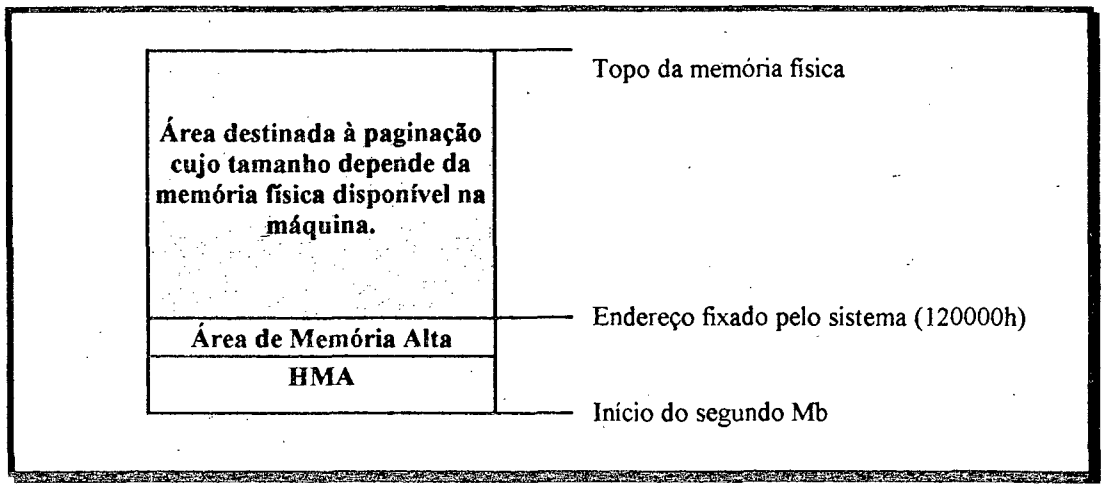

Fig. 4.17 - Detalhes da inicialização da área da memória de paginação.

Observe que, como as estruturas de paginação iniciais mapeavam todo o espaço endereçável de maneira que os endereços lineares e físicos eram idênticos, e como só alteramos esse mapeamento no trecho ocupado pelo processo chamador, o restante fica igual. Essa igualdade permite que acessemos as páginas da área da memória de paginação sem ter que guardar seus endereços lineares. Note, ainda, que as páginas ocupadas ficam com dois endereços lineares apontando para o mesmo endereço físico, enquanto que as páginas livres tem apenas o endereço linear inicial.

O conteúdo do registrador CR3 é ajustado para usar o novo ambiente de endereçamento e os serviços do DOS são usados para liberar a área ocupada pelo programa cujo processo chamou a função Background como se o mesmo houvesse terminado.

A alocação de páginas na memória de paginação é feita por um função específica. Não existe função de desalocação porque partimos do pressuposto de que tudo o que se instalar na memória alta é para ficar sempre disponível ao sistema físico gerenciado.

4.6 Interrupções e Exceções

O tratamento de interrupções é um dos aspectos mais importantes na operação do computador no modo virtual. Isso se deve à troca do modo de operação do modo virtual para o modo protegido feita automaticamente pelo processador ao ocorrer uma interrupção. Essa troca de modo de operação não pode ser evitada, pois é definida pelo projeto do processador.

O mecanismo de atendimento das interrupções inicia pela ocorrência da interrupção no modo virtual. O processador troca de modo de operação passando para o modo protegido. No modo protegido, usa o registrador IDTR para localizar a IDT. Usando o número da interrupção como índice, localiza o descritor da IDT que endereça e define o tipo de rotina de tratamento que está instalada para a manipulação da interrupção ocorrida e vai executá-la, seja por troca de tarefa, no caso de 'task gate' ou, mantendo o ambiente da tarefa ativa, como no caso de 'interrupt gate'.

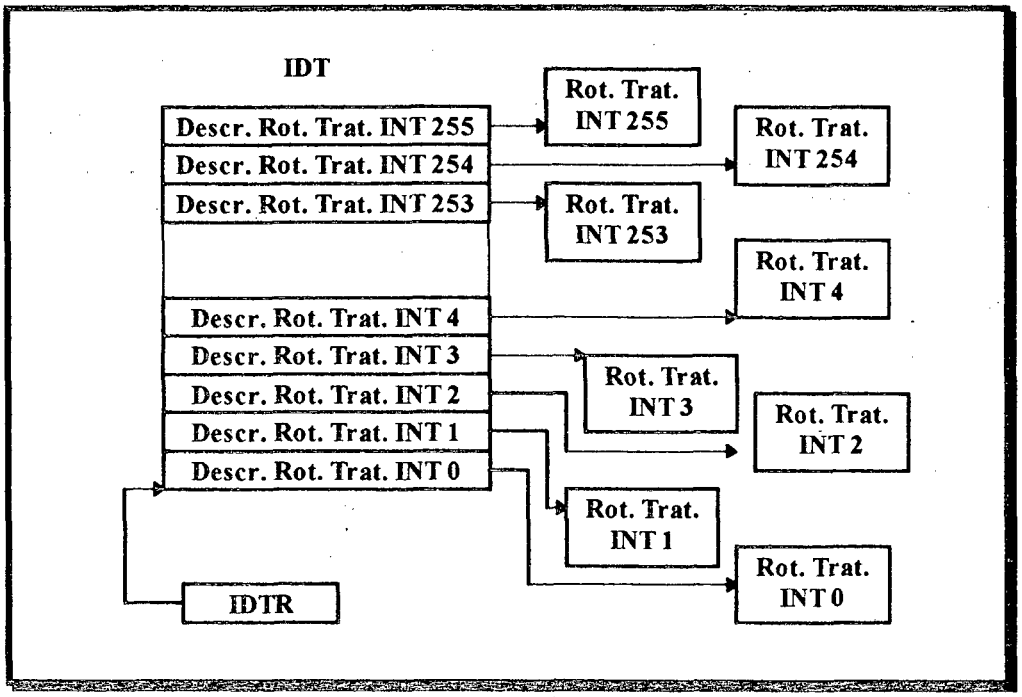

Fig. 4.18 - Esquema de tratamento de interrupções no modo protegido.

No caso de tratamento de interrupções no modo virtual, nesta implementação, foram usados 'interrupt gates' e dirigiu-se o tratamento de todas as interrupções para um código comum a todas elas com função de retornar ao modo virtual e tratar as interrupções com o uso das rotinas normais do DOS e do BIOS. A esse código chamamos 'Monitor Virtual'.

4.6.1 Monitor Virtual

O Monitor Virtual tem, como já citado, a tarefa de redirecionar o tratamento de interrupções ao DOS e ao BIOS. A estrutura básica do tratamento de interrupções consta da IDT endereçando para pequenas rotinas de interrupção, cadastradas como 'interrupt gates' que são constituídas de uma instrução PUSH para colocar na pilha o número da interrupção e um salto, através de uma instrução JMP, para seguir a execução no código do Monitor Virtual.

A pilha de nível zero do processo interrompido já continha as informações colocadas lá pelo processador durante a interrupção com troca de nível de privilégio. A estas informações, agregamos o número da interrupção, obtendo o perfil da pilha visto na Fig. 4.18.

A maneira normal de tratar as interrupções consiste em tirar da pilha de nível zero (modo protegido) o endereço onde ocorreu a interrupção (CS e IP) e o conteúdo do registrador FLAGS e colocá-los, abrindo espaço para eles (seis bytes) na pilha de nível tres (modo virtual). Na pilha de nível zero, no lugar do endereço onde ocorreu a interrupção, colocamos o endereço da rotina de atendimento da interrupção do DOS, retirado do vetor de interrupção do DOS. A Fig. 4.19 ilustra as duas situações das duas pilhas antes e depois da manipulação descrita acima.

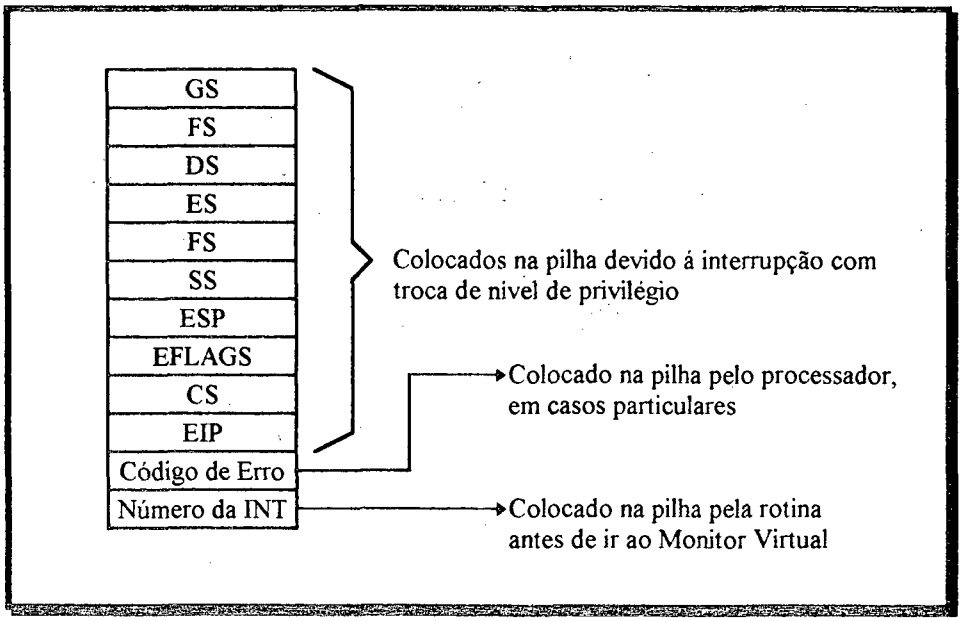

Fig. 4.19 - Conteúdo da pilha na entrada do Monitor Virtual.

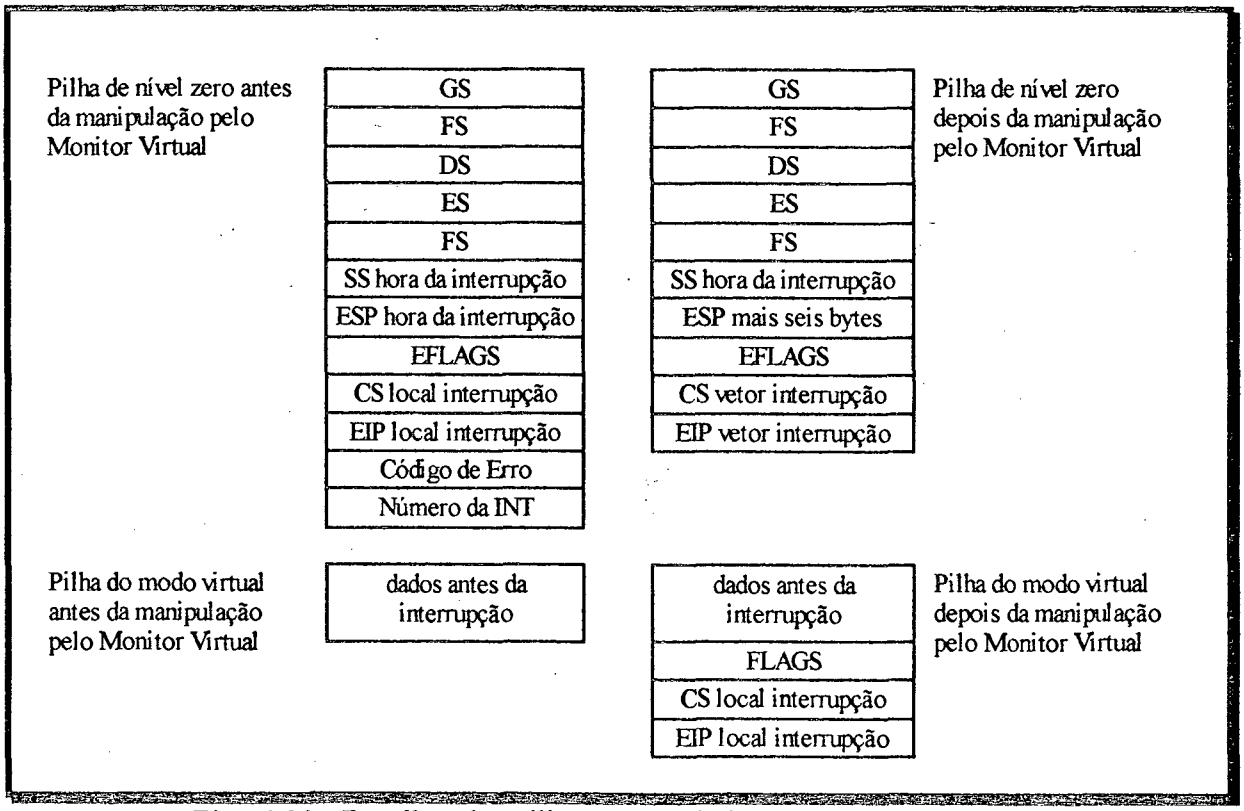

Fig. 4.20 - Detalhes das pilhas manipuladas pelo Monitor Virtual.

Tendo efetuado a manipulação das pilhas, ao executarmos uma instrução IRET, o processador vai para a rotina de tratamento do DOS e, ao final desta, ao executar um outro IRET, volta ao ponto onde tinha ocorrido a interrupção, conforme podemos verificar na Fig. 4.20.

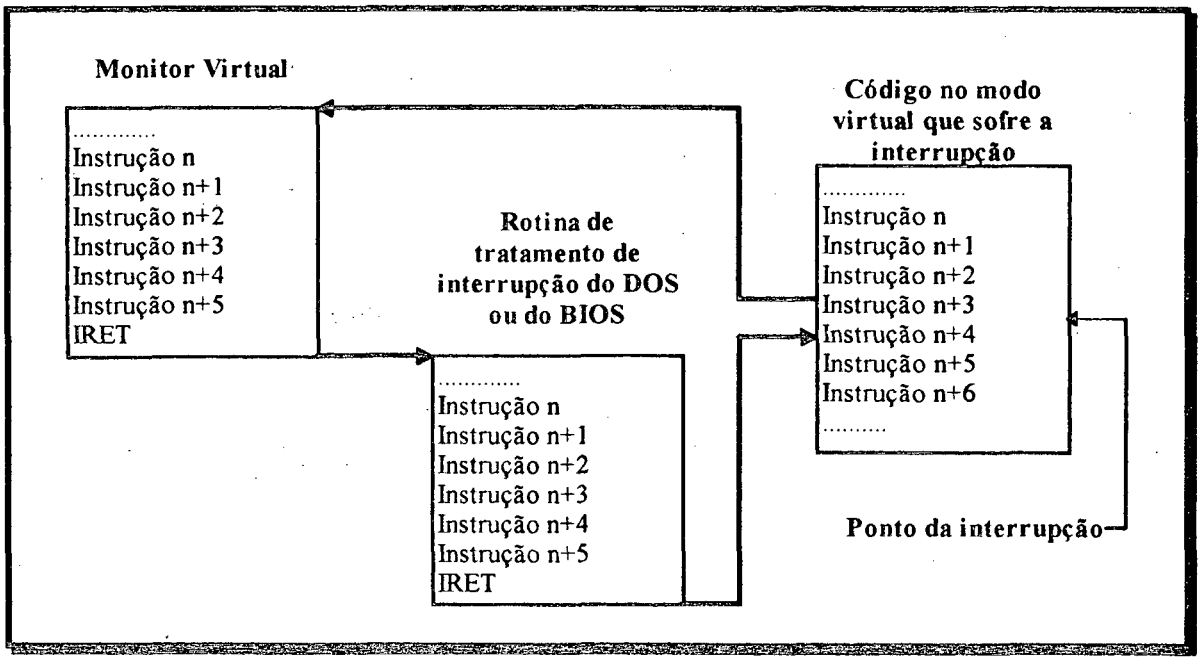

Fig. 4.21 - Seqüência do processamento no tratamento das interrupções.

O Monitor Virtual trata a maioria das interrupções de maneira idêntica, através do procedimento acima descrito. Existem algumas situações, porém, em que esta sequência é obrigada a sofrer desvios. Essas variações à regra, do ponto de vista do Monitor Virtual, são:

- **Interrupção do relógio (INT8)**, quando o Monitor Virtual prepara o sistema para retornar ao modo protegido após tratar a interrupção pelo BIOS para executar o tratamento dos relógios do sistema e o escalonador. O retorno é obtido chamando uma interrupção de software conhecida.
- **Interrupção de retorno ao modo protegido (INT61)**, quando, após tratar a interrupção do relógio do computador pelo manipulador do BIOS, a execução retornou ao modo protegido para que o Monitor Virtual inicie o tratamento dos relógios do sistema. Isso pode implicar em nova ida ao modo virtual para executar uma rotina de interrupção associada ao relógio, causando a necessidade de novo retorno ao modo protegido para continuar o tratamento destes relógios. Isso é feito com outra interrupção de software conhecida.
- **Interrupção de retorno ao modo protegido (INT62)**, quando, após executar a rotina de interrupção associada ao relógio do sistema, a execução retornou ao Monitor Virtual para a continuação do tratamento dos relógios do sistema.
- **Interrupção de serviço (INT60)** da interface do sistema com as aplicações, quando o Monitor Virtual desvia para as rotinas de serviço.
- **Exceções ocorridas no modo protegido**, geralmente por falha no código do sistema, quando o Monitor Virtual coloca, na tela, informações necessárias para depuração e paraliza o processador.

No caso da interrupção do relógio (INT8), o Monitor Virtual deve preparar o sistema para retornar ao modo protegido após executar a rotina de tratamento da interrupção pelo BIOS. O retorno se deve à necessidade de executar o tratamento dos relógios do sistema e de executar o escalonador.

O escalonador, por ser preemptivo, deve executar sempre após uma interrupção de hardware. Ao ocorrer uma interrupção de hardware qualquer, a execução desvia do código que estava sendo executado para o Monitor Virual, que assume o controle, podendo desviar a execução para qualquer lugar. Em nosso caso, foi definida a interrupção do relógio que o Monitor Virtual desvie para o escalonador.

Como a única maneira de passar do modo virtual para o modo protegido é através de uma interrupção ou exceção, o retorno é obtido chamando uma interrupção de software que seja reconhecida pelo sistema como um pedido para voltar ao modo protegido. Para isso deixamos residente no modo virtual uma pequena função que possui apenas uma chamada à interrupção de retorno ao modo protegido.

No caso da interrupção do relógio do computador, pelos motivos descritos acima, acrescentamos um pequeno desvio ao tratamento da interrupção de maneira que, após a execução da rotina de tratamento do BIOS, ao invés de retornar para o código interrompido, executamos essa pequena função que aciona uma interrupção conhecida e voltamos ao modo protegido. Para podermos usar o endereço desta função especial na pilha do modo virtual, guardamos o endereço de retorno ao código interrompido em uma variável do sistema.

Ao retornar ao modo protegido, o Monitor Virtual identifica a situação que está ocorrendo, através da interrupção, e recupera o endereço anteriormente guardado, de retorno ao código interrompido, e desvia para o tratamento dos relógios. A figura Fig. 4.21 ilustra este desvio.

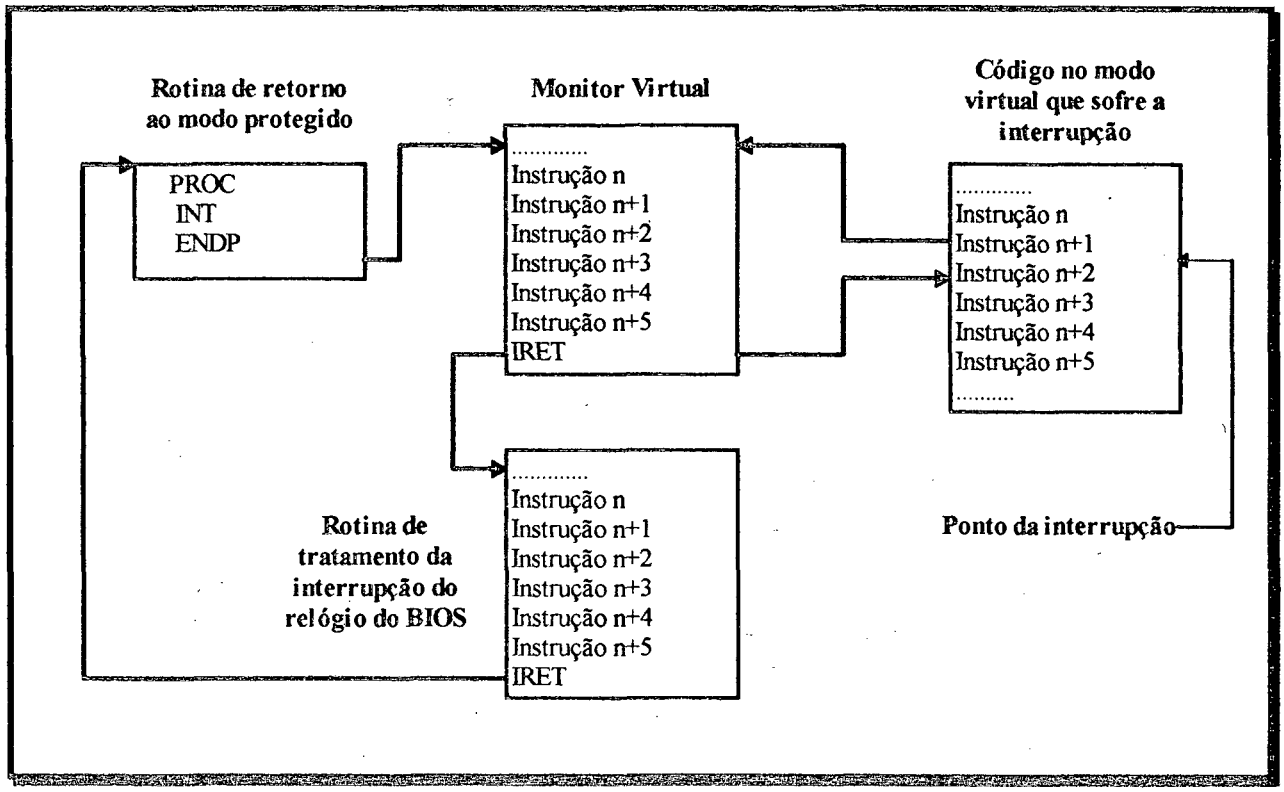

Fig. 4.22 - Sequência do processamento no caso do tratamento da interrupção do relógio.

Cabe ressaltar que o Monitor Virtual é acionado duas vezes no tratamento da interrupção do relógio. A primeira vez pela interrupção de hardware acionada pelo relógio e a segunda vez pela interrupção de software que pede o retorno ao modo protegido.

Outro tratamento que foge da rotina é o caso da interrupção de retorno ao modo protegido após executar a rotina de interrupção associada ao relógio do sistema. Os relógios do sistema tem por função acionar uma rotina escrita ao estilo das rotinas de interrupção. O tratamento dos relógios do sistema consiste em decrementar um contador a cada interrupção do relógio de hardware (cada 55 ms) e, ao chegar o instante de disparo do relógio, quando o contador chegar a zero, o processador deve ir ao modo virtual e executar esta rotina associada ao relógio que está sendo tratado. O processo é idêntico ao tratamento da interrupção do relógio de hardware, com a exceção de que a rotina de tratamento é diferente para cada relógio e o ponto de retorno no modo protegido é diferente do ponto de retorno quando do tratamento da interrupção do relógio de hardware. Portanto a rotina que pede para retornar ao modo protegido para continuar a tratar os demais relógios do sistema deve acionar outra interrupção para que o Monitor Virtual possa identificá-la. A Fig. 4.22 ilustra esse tratamento.

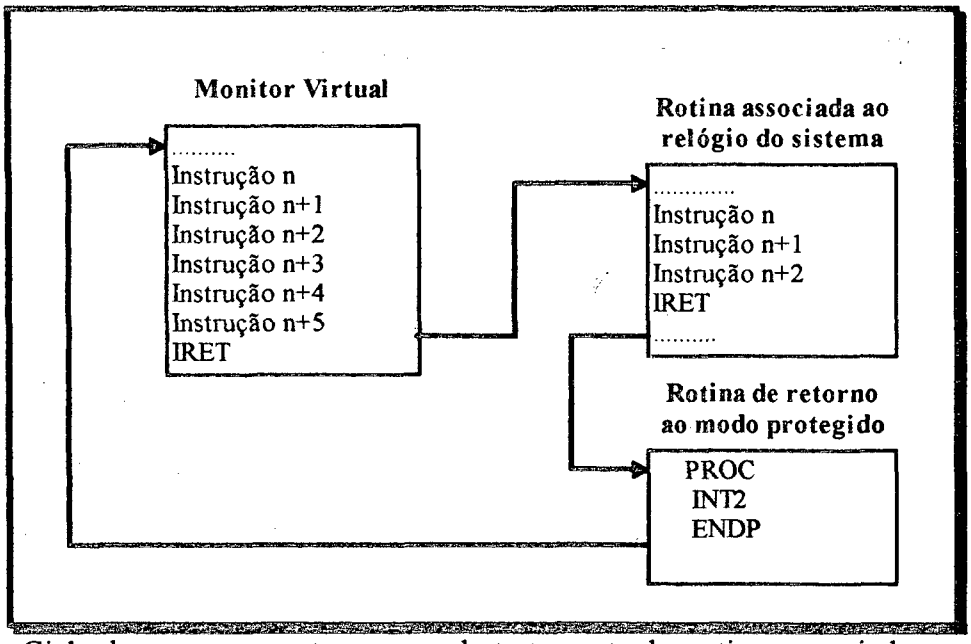

Fig. 4.23 - Ciclo do processamento no caso do tratamento das rotinas associadas aos relógios.

A sequência do tratamento de relógios do sistema segue as seguintes etapas:

[1] Durante a execução de um processo no modo virtual, ocorre uma interrupção do relógio de hardware e o processador troca para o modo protegido indo executar o Monitor Virtual.

[2] Identificada a interrupção como sendo do relógio de hardware, o Monitor Virtual redireciona o processamento para a rotina de atendimento do BIOS, preparando o retorno ao modo protegido.

[3] Após executada a rotina de atendimento do BIOS, o endereço colocado pelo Monitor Virtual na pilha do modo virtual desvia a execução para a rotina de retorno ao modo protegido após tratamento da interrupção do relógio de hardware.

[4] Essa rotina de retorno ao modo protegido chama uma interrupção de software conhecida (INT61) e retorna ao Monitor Virtual.

[5] O Monitor Virtual, identificando a interrupção como pedido de retorno ao modo protegido após o tratamento de uma interrupção do relógio de hardware,

recupera o endereço que aponta para o ponto de interrupção do processo do modo virtual e salta para o trecho de tratamento dos relógios do sistema.

[6] Durante o tratamento dos relógios do sistema, um ou mais relógios podem chegar ao momento de executar a sua rotina associada. Quando isso acontece, o tratamento dos relógios do sistema salva o endereço de retorno ao modo virtual e prepara as pilhas do modo virtual e do modo protegido para que a execução passe para a rotina associada ao relógio e retorne ao modo protegido.

[7] Depois de executar a rotina associada ao relógio do sistema, o endereço na pilha do modo virtual leva à execução de uma rotina de retorno ao modo protegido.

[8] A rotina de retorno ao modo protegido, usando uma interrupção (INT62), conhecida pelo Monitor Virtual, causa a troca de modo de operação do processador e a volta ao Monitor Virtual.

[9] O Monitor Virtual, identificando a interrupção, recupera os endereços guardados e salta para o ponto adequado para continuar o tratamento dos relógios do sistema que ainda estiverem por tratar.

[10] As etapas [6] a [9] podem se repetir tantas vezes quantas necessário para tratar todos os relógios que tiverem atingido seu ponto de disparo.

[11] Após tratar todos os relógios do sistema, o Monitor Virtual verifica se é hora de executar o Escalonador e o faz, se for necessário. Se não for necessário executar o escalonador, o Monitor Virtual retorna para o ponto de interrupção do processo no modo virtual. O Escalonador, se chamado, escolhe o processo de mais alta prioridade entre os que estão no estado PRONTO e troca de tarefa indo executá-lo, se for um processo diferente que o que estava executando, ou executa um retorno ao Monitor Virtual para voltar ao mesmo processo anterior..

O próximo tratamento que foge à rotina do Monitor Virtual é o caso da aso interrupção dos serviços da interface do sistema com as aplicações. Quando o Monitor Virtual identifica a interrupção como sendo a dos serviços de interface, desvia para as rotinas de serviço, de acordo com o serviço solicitado. Esses serviços serão objeto de detalhamento adiante.

Essas rotinas, em número variável, atualmente cerca de trinta, tem seus endereços agrupados em um vetor que é utilizado por uma rotina única de distribuição. O Monitor Virtual desvia para essa rotina através de uma instrução JMP e ela, após identificar o endereço da função desejada, executa outro JMP para a rotina. Desta maneira, todas as funções de interface de serviços do sistema são executadas como se fizessem parte do Monitor Virtual. Isso significa, entre outras coisas, que as funções da interface do sistema executam no modo protegido.

Resta a verificar apenas o caso das exceções ocorridas no modo protegido, geralmente por falha no código do sistema, quando o Monitor Virtual põe na tela informações necessárias para depuração e paraliza o processador. Neste caso, a análise dos dados expostos permite identificar o problema para correção. Essa situação não deve ocorrer em operação normal.

5. Interface com as aplicações

Todo o sistema operacional precisa fornecer aos aplicativos alguma maneira de acessar seus serviços. Isso é feito através das funções de interface. O DOS fornece seus serviços através das funções da INT21, o BIOS do sistema de vídeo fornece seus serviços através da INT10, e assim por diante. Esta implementação possui serviços na área de semáforos, mensagens, gerência de processos, entre outros, que não existem no DOS e que precisam ser colocados à disposição dos aplicativos. Para isso, foi definida a INT60 como interrupção de acesso e foi criada uma biblioteca de funções na linguagem C para acessar os serviços do sistema.

5.1 Definições de Dados

Nesse item definiremos algumas constantes usadas pelo sistema, com a intenção de padronizar as comparações boleanas.

5.1.1 Constantes

- #define **TRUE** 1 - Definição da constante lógica VERDADEIRO.
- #define **FALSE** 0 - Definição da constante lógica FALSO.

5.1.2 Definição de Tipos de Variáveis

Com o objetivo de padronizar os tipos de variáveis da interface com nomes conhecidos na área dos sistema multitarefa para tempo real, foram criados tipos de variáveis particulares ao sistema. Esses tipos e estruturas de dados estão definidos aqui.

- #define **process** void - Processo rodando sob o XDOS
- #define **temporiz** interrupt far - Rotina temporizada associada a um relógio do sistema
- typedef unsigned char **pid** - Identificador de processo no XDOS
- typedef unsigned char **relógio** - Relógio devolvido pelo XDOS
- typedef unsigned **semáforo** - Semáforo
- typedef unsigned **word** - Palavra
- typedef unsigned char **byte** - Byte
- typedef void far * **buffer** - Mensagens trocadas pelos processos
- typedef char * **string** - Cadeia de caracteres
- typedef struct { - Estrutura para o tempo do sistema

  int ANO;

  char MES;

char DIA;

char SEMANA;

char NUSO;

char HORA;

char MINUTO;

char SEGUNDO;

char CENTS; } **tempo**;

5.2 - Definição de Chamadas de Serviços da Interface do Sistema

As funções da interface foram definidas, pela arquitetura do sistema, de acordo com as necessidades de um sistema multitarefa para tempo real e foram agrupadas aqui de acordo com a área de atuação.

5.2.1 - Serviços de Gerenciamento de Processos

5.2.1.1 Ativa

Função : Coloca uma rotina a executar como um processo concorrente. Essa rotina instala os processos com prioridade zero. Foi mantida por compatibilidade com softwares existentes de versões anteriores. A rotina Instala deve ser usada preferencialmente no lugar desta. Outra diferença entre essa função e a função Instala é a posição onde a pilha do processo é alocada. Essa função aloca a pilha na área de memória baixa, através de uma chamada do serviço de gerenciamento de memória. A função Instala aloca a pilha através de uma chamada de função do DOS, na memória NEAR, no mesmo segmento onde estão os dados e a pilha do programa.

Chamada: pid **Ativa**(process *,string,unsigned)

Parâmetros:

[1] Ponteiro para a rotina que será instalada como um processo.

[2] Identificação externa mnemônica do processo constituída de quatro caracteres.

[3] Tamanho da pilha a ser criada para o processo.

Retorno: Devolve o identificador interno numérico do processo. Se o valor devolvido for igual a 0, houve erro na instalação do processo.

5.2.1.2 Instala

Função: Coloca uma rotina a executar como um processo concorrente, com prioridade.

Chamada: pid **Instala**(process *,string,unsigned,unsigned)

Parâmetros:

[1] Ponteiro para a rotina que será instalada como um processo

[2] Identificação externa mnemônica do processo constituída de quatro caracteres.

[3] Tamanho da pilha a ser criada para o processo.

[4] Prioridade entre 0 e 9, quanto maior o valor, mais alta é a prioridade.

Retorno: Devolve o identificador interno numérico do processo. Se o valor devolvido for igual a 0, houve erro na instalação do processo.

5.2.1.3 Termina

Função: Excluir o processo ativo da estrutura de execução concorrente.

Chamada: void **Termina**(void)

Parâmetros: Essa função não tem parâmetros.

Retorno: Essa função não tem retorno.

5.2.1.4 TrocProc

Função: Libera o processador para outro processo rodar. O processo ativo continua na estrutura de gerenciamento de processos, pronto para executar.

Chamada: void **TrocProc**(void)

Parâmetros: Essa função não tem parâmetros.

Retorno: Essa função não tem retorno.

5.2.1.5 Identif

Função: Devolve o identificador interno numérico do processo correspondente ao mnemônico do mesmo.

Chamada: pid **Identif**(string)

Parâmetros:

[1] Cadeia de quatro caracteres com o identificador mnemônico do processo a identificar.

Retorno: Retorna o identificador do processo ativo. Se o retorno for igual a zero, o mnemônico não pode ser encontrado entre os processos inicializados no sistema.

5.2.1.6 Mnemônico

Função: Devolve o identificador externo mnemônico do processo a partir do identificador interno numérico.

Chamada: int **Mnemônico**(pid,string)

Parâmetros:

[1] Identificador interno numérico do processo cujo identificador mnemónico se procura.

[2] Cadeia de quatro caracteres para que o sistema coloque nele o identificador mnemônico do processo.

Retorno:

[1] Se o retorno for igual a zero, obtivemos sucesso e cadeia de caracteres fornecida foi preenchida com o identificador buscado.

[2] Se o retorno for diferente de zero, o identificador interno numérico do processo que foi fornecido é invalido e a cadeia de caracteres fornecida não foi preenchida.

5.2.1.7 ProcAtiv

Função: Essa função busca nas variáveis do gerenciador de processos, o identificador do processo que está ativo. Logicamente esta função sempre retornará ao processo o seu próprio identificador, pois só pode ser executada pelo processo que estiver sendo executado.

Chamada: pid **ProcAtiv**(void)

Parâmetros: Essa função não tem parâmetros.

Retorno: Retorna o identificador do processo ativo.

5.2.2 - Serviços de Gerenciamento de Semáforos e Mensagens

5.2.2.1 InicS

Função: Inicializa os dados da estrutura de uma variável do tipo semáforo.

Chamada: void **InicS**(semáforo far*,char)

Parâmetros:

[1] Ponteiro para a variável, do tipo semáforo, definida pela aplicação que está sendo executada, que será inicializada.

[2] Valor inicial a ser colocado no contador da estrutura da variável do tipo semáforo que está sendo inicializada. Tem como significado o número de recursos disponíveis nesse semáforo ou o número de processos que poderão entrar concomitantemente no trecho protegido pelo semáforo.

Retorno: Essa função não tem retorno.

5.2.2.2 P

Função: Realiza uma operação P sobre uma variável do tipo semáforo. Essa operação decrementa o contador do semáforo e testa o resultado. Se o resultado for negativo, bloqueia o processo e o coloca no fim da fila deste semáforo. Se o resultado for zero ou positivo, permite que o processo continue a sua execução.

Chamada: void **P**(semáforo far*)

Parâmetros:

[1] Ponteiro para a variável semáforo que sofrerá a operação P.

Retorno: Essa função não tem retorno.

5.2.2.3 V

Função: Realiza uma operação V sobre uma variável do tipo semáforo. Essa operação incrementa o contador do semáforo e, se o resultado for zero ou negativo, não faz nada. Se o resultado for positivo, desbloqueia o processo que estiver no início da fila deste semáforo.

Chamada: void **V**(semáforo far*)

Parâmetros:

[1] Ponteiro para a variável semáforo que sofrerá a operação V.

Retorno: Essa função não tem retorno.

5.2.2.4 Send

Função: Envia uma mensagem para um outro processo qualquer.

Chamada: char **Send**(pid,buffer)

Parâmetros:

[1] Identificador interno numérico do processo destino.

[2] Ponteiro para o buffer de mensagem.

Retorno: Devolve: 0 se houve sucesso.

1 se não houver espaço para novo bloco de memória do sistema

2 se o processo destino for inválido

5.2.2.5 WaitT

Função: Espera por uma mensagem enviada por um processo qualquer.

Chamada: buffer **WaitT**(unsigned)

Parâmetros:

[1] Tempo de espera, em unidades de 55 ms, durante o qual o processo permanecerá bloqueado aguardando a mensagem. Se o tempo de espera for igual a zero, o processo esperará indefinidamente até chegar a mensagem. Se jamais chegar mensagem, o processo jamais será desbloqueado. Essa função forma conjunto e deve ser sempre empregada com a função Send.

Retorno: Devolve o ponteiro (far) para a mensagem ou NULL se estourou o tempo.

5.2.2.6 Wake

Função: Coloca uma mensagem no inicio da fila de mensagens de um processo, desbloqueando-o se ele estiver esperando mensagem especificamente do processo ativo. O processo que enviou a mensagem continua executando e o que recebeu passa ao estado PRONTO e será executado quando o escalonador o escolher. Essa função forma conjunto e deve ser sempre empregada junto com a função Sleep.

Chamada: char **Wake**(pid,buffer)

Parâmetros:

[1] Identificador do processo receptor.

[2] Ponteiro para o buffer de mensagem.

Retorno: Retorna :

0 - se houve sucesso.

1 - se não houver espaço para novo bloco de memória do sistema

2 - se o processo destino for inválido

3 - se o processo receptor não estiver bloqueado esperando a chegada de uma mensagem especificamente do processo ativo.

5.2.2.7 Sleep

Função: Espera por uma mensagem de um processo especifico.

Chamada: buffer **Sleep**(pid,unsigned)

Parâmetros:

[1] Identificador do processo destino.

[2] Tempo de espera, em unidades de 55 ms, durante o qual o processo permanecerá bloqueado aguardando a mensagem. Se o tempo de espera for igual a zero, o processo esperará indefinidamente até chegar a mensagem. Se jamais chegar mensagem, o processo jamais será desbloqueado.

Retorno: Devolve o ponteiro (far) para a mensagem recebida, ou NULL se estourou o tempo de espera.

5.2.3 - Serviços de Gerenciamento de Memória

Essas funções se referem ao gerenciador de memória baixa, na área de endereçamento linear acessível ao DOS e acessível, portanto, aos processos executando no modo virtual. As demais áreas de memória gerenciadas pelo sistema não possuem funções na interface dos serviços com as aplicações porque estas aplicações, executando no modo virtual, não podem gerar os endereços lineares necessários para acessá-las.

5.2.3.1 LibBuf

Função: Devolve um bloco de memória para o gerenciador de memória baixa.

Chamada: char **LibBuf**(buffer)

Parâmetros:

[1] Ponteiro para um bloco de memória anteriormente recebido com uma mensagem pela execução da função WaitT ou recebido pela execução da função AlocBuf.

Retorno: Retorna zero se houve sucesso, se o retorno não for zero, o ponteiro para o bloco de memória é inválido e o bloco de memória indicado pelo ponteiro não foi reconhecido como um bloco de memória válido.

5.2.3.2 AlocBuf

Função: Aloca um bloco de memória do gerenciador de memória baixa.

Chamada: buffer **AlocBuf**(int)

Parâmetros:

[1] Tamanho do bloco de memória a ser alocado, em bytes.

Retorno: Devolve o endereço do bloco de memória ou NULL se não houver memória livre suficiente para acomodar o bloco solicitado.

5.2.4 - Serviços de Gerenciamento de Temporização

5.2.4.1 CriaRel

Função: Coloca uma função na estrutura de execução temporizada, alocando um relógio do sistema para controlar o tempo.

Chamada: relógio **CriaRel**(void far *,int)

Parâmetros:

[1] Ponteiro para uma função definida como 'interrupt far'.

[2] Parâmetro a ser passado para a função, por ocasião de sua execução.

Retorno: Devolve o número do relógio alocado. Se devolver um valor igual a 0FFH, não existe relógio do sistema disponível para alocação.

5.2.4.2 ParteRel

Função: Inicializa o campo do contador do relógio, iniciando a contagem do tempo para execução da função. A cada interrupção do relógio de hardware, o contador é decrementado. Ao chegar a zero, a função associada ao relógio do sistema é executada.

Chamada: void **ParteRel**(relogio,unsigned)

Parâmetros:

[1] Número de um relógio anteriormente criado.

[2] Tempo para execução da função, em unidades de 55 ms.

Retorno: Esta função não tem retorno.

5.2.4.3 ParaRel

Função: Para definitivamente a contagem atual do tempo, colocando zero no contador sem executar a função associada. Com outra chamada à função ParteRelog, o contador será reinicializado e a contagem será reiniciada do princípio.

Chamada: void **ParaRel**(relógio)

Parâmetros:

[1] Número de um relógio anteriormente criado.

Retorno: Esta função não tem retorno.

5.2.4.4 CongRel

Função: Essa função suspende temporariamente a contagem do tempo. A contagem será reiniciada do ponto onde parou com uma chamada subsequente à função DescRelog.

Chamada: void **CongRel**(relógio)

Parâmetros:

[1] Número de um relógio anteriormente criado que esteja com a contagem de tempo em andamento.

Retorno: Esta função não tem retorno.

5.2.4.5 DescRel

Função: Essa função reinicia a contagem do tempo do relógio, do ponto em que havia sido suspensa pela função CongRel.

Chamada: void **DescRel**(relógio)

Parâmetros:

[1] Número de um relógio anteriormente criado cuja contagem de tempo tenha sido suspensa com a função CongRel.

Retorno: Esta função não tem retorno.

5.2.4.6 LibeRel

Função: Exclui a função da estrutura de execução temporizada, devolvendo o relógio que estava sendo usado por ela ao sistema.

Chamada: void **LibeRel**(relógio)

Parâmetros:

[1] Número de um relógio anteriormente criado.

Retorno: Esta função não tem retorno.

5.2.5 - Outros Serviços

5.2.5.1 MonoBeg

Função: Desativa o sistema de execução de processos concorrentes. A execução volta a ser monotarefa, até que o sistema seja restabelecido através de uma chamada à função MonoEnd. No sistema tudo continua igual com exceção do escalonador que fica desabilitado. Essa função foi colocada na interface do sistema para fins de depuração, não devendo ser utilizada em situação normal.

Chamada: void **MonoBeg**(void)

Parâmetros: Essa função não tem parâmetros.

Retorno: Essa função não tem retorno.

5.2.5.2 MonoEnd

Função: Reativa o sistema de execução de processos concorrentes, após ter sido suspenso com o uso da função MonoBeg.

Chamada: void **MonoEnd**(void)

Parâmetros: Essa função não tem parâmetros.

Retorno: Essa função não tem retorno.

5.2.5.3 BackGrnd

Função: Mantém residentes os processos ativados pelo programa em execução, ocupando a área de memória definida na carga do programa executável. Essa função ativa os mecanismos de paginação para liberar a área de memória dos programas transitórios para que se possa instalar mais processos. Isso feito, retorna a executar o interpretador de comandos como um processo concorrente, permitindo acesso ao sistema DOS e, consequentemente, permitindo a carga de mais processos. Quando chamada, aciona os mecanismos de paginação já inicializados, liberando a memória ocupada pelos processos e mantendo-os residentes em outra parte da memória de paginação.

Chamada: void **Backgrnd**(void)

Parâmetros: Essa função não tem parâmetros.

Retorno: Essa função não tem retorno.

5.2.5.4 Residente

Função: Essa função é usada para deixar residentes processos já instalados e que não devem ser utilizados através da estrutura de paginação. Deve ser empregada para instalar processos auxiliares do sistema como, por exemplo, 'device drivers', 'drivers' de rede e rotinas de atendimento à interrupções, entre outros. Tudo, enfim, aquilo que for auxiliar do sistema e que será compartilhado por todos os processos deverá ser instalado usando essa função.

Chamada: void **Residente**(void)

Parâmetros: Essa função não tem parâmetros.

Retorno: Essa função não tem retorno.

6. Conclusões e Sugestões

Iniciou-se a execução deste trabalho pela pesquisa, na bibliografia disponível, dos assuntos que julgamos relevantes, como a arquitetura dos processadores 80386 e 80486, principalmente no que tange aos modos de operação dos mesmos e as características de sistemas operacionais para tempo real.

Após esta fase inicial, desenvolveu-se um protótipo do sistema que foi projetado, o qual se encontra operacional em todos os detalhes descritos neste trabalho. Atualmente envida-se esforços para incluir primitivas de acesso a redes de computadores tanto via saidas seriais como via placas ethernet. Esse acréscimo se encontra em fase de testes e depuração.

Neste ítem incluiu-se as conclusões a que se chegou após terem sido pesquisados todos os assuntos que foram considerados necessários para esse projeto e ter sido implementado um protótipo do sistema que foi projetado para fins de teste das premissas. Incluiu-se, também, as sugestões que foram consideradas interessantes para melhorar o sistema stualmente implementado no protótipo e descrito nos ítems anteriores.

6.1 Conclusões

Concluiu-se que é possível implementar um sistema operacional multitarefa para tempo real usando o modo virtual 86 dos processadores 80386 e 80486 para permitir o uso da memória paginada.

Conseguiu-se, com o uso do modo virtual 8086, ultrapassar o limite de 640 Kb que o sistema operacional DOS impõe para os programas que executa.

Mostrou-se, através de revisão da literatura, os princípios de programação do modo protegido, o que pode ser bastante detalhado com o estudo cuidadoso dos códigos fonte que acompanham este trabalho.

Criou-se um protótipo de sistema operacional multitarefa para tempo real que usa o modo virtual 8086 para executar tarefas desenvolvidas como se fossem simples programas DOS, que se está habituado a desenvolver no dia-a-dia.

Detalharam-se as técnicas de uso da paginação para trocar as tarefas virtuais de lugar para que outras possam usar este mesmo espaço de endereçamento linear e, assim, enganar o DOS.

6.2 Sugestões de Otimização e Expansão do Sistema

Este trabalho não é completo e nunca teve a pretensão de chegar a sê-lo. Algumas características foram escolhidas de forma a dar condições de fácil compreensão, clareza e estruturação. Além disso, outras características foram excluídas pelos mesmos motivos. Portanto, muito pode ser melhorado e adicionado ao presente sistema.

Sugere-se, a seguir, pontos onde se pode expandir o sistema atual de maneira a torná-lo mais eficiente e completo.

6.2.1 Sugestões de Otimização

Estas são as sugestões para uma otimização do sistema atual:

- Alterar o Monitor Virtual para que o mesmo possa ser executado com as interrupções habilitadas. Isso dará maior controle no que tange ao tempo de atendimento das interrupções. Para isso, é necessário que o Monitor Virtual reconheça o modo em que as interrupções estão acontecendo, cuidando da ida ao modo virtual e retorno ao modo protegido de maneira a ir a qualquer posição onde a execução tenha sido interrompida. Atualmente, o retorno só pode ser feito para duas posições no Monitor Virtual: o ponto de início do tratamento dos relógios do sistema e o ponto de sequência do mesmo tratamento.
- Tendo sido desenvolvida a otimização sugerida acima, é possível um estudo detalhado das funções de interface para que elas executem com as interrupções habilitadas, quando possível, pois as mesmas executam como se fizessem parte do monitor virtual.
- Desenvolver rotinas de tratamento de interrupções que executem no modo protegido. Isso significa maior rapidez no tratamento dessas interrupções. No caso da interrupção do relógio de hardware (INT 8), por não ser mais necessário a ida ao modo virtual para tratá-la, em que se deve retornar ao modo protegido para outras providências, isso é altamente desejável, pois a troca de modo usa muitos ciclos de máquina para ser efetivada.

6.2.2 Sugestões de Expansão

Estas sugestões são de caráter de expansão do sistema atual, visando acrescentar características que o sistema não possui atualmente:

- Permitir a execução de tarefas no modo protegido. Para tanto, deve-se criar mecanismos de carga destas tarefas e de instalação dentro do sistema. É possível ao sistema, da maneira como está estruturado atualmente, executar tarefas no modo virtual e tarefas no modo protegido ao mesmo tempo.
- Acrescentar 'device drivers' de rede e criar funções colocadas na interface para permitir que tarefas executando em máquinas diferentes troquem mensagens. Isso aumentaria grandemente o potencial de uso deste sistema em automação industrial.

7. Bibliografia

[1]BARRON, Tovey. *Protected-mode Debugging Using In-circuit Emulators - Making a case for emulation.* Dr. Dobb's Journal, February 1992.

[2]COMER, Douglas. *Operating Systems Design Vol 1: The XINU Approach (PC Edition).* Prentice-Hall, 1988.

[3]DIAS, Wilson A. *8086 8088 Hardware - Software - Aplicações - Projetos.* McGraw Hill, 1990.

[4]*DOS Protected Mode Interface (DPMI) - Protected Mode API For DOS Extended Applications* - Version 0.9. 1990.

[5]DUNCAN, Ray et all. *Ampliando o DOS.* Ciência Moderna, 1991.

[6]DUNCAN, Ray. *MSDOS Avançado - Guia do Usuário.* Makron Books, 1990.

[7]ECKHARDT, V. et all. *Die Grenzen liegen jetzt bei der Mechanik.* Elektronik, Setembro 1993.

[8]FERRI, Enrique H. H. *Introdução 80386/486 - Arquitetura e Componentes de Hardware.* Erica, 1990.

[9]FRIED, Stephen. *Accessing Hardware from 80386/486 Protected Mode Part I - Understanding the 386 architecture may simply be a matter of building on what you already know.* Dr. Dobb's Journal, May 1990.

[10]FRIED, Stephen. *Accessing Hardware from 80386/486 Protected Mode Part II - A 4-gigabyte memory model and features that control 80386/486 paging make FAR pointers obsolete.* Dr. Dobb's Journal, June 1990.

[11]GREEN, Tom. *80386/486 Protected Mode Multitasking - Putting the 386 to work under MS-DOS.* Dr. Dobb's Journal, September 1989.

[12]HAPPACHER ,M. *Dem japanischen Vorstoβ gewachsen?.* Elektronik, Setembro 1993.

[13]HARALD, A. *MSDOS in a box.* Revista C't, Heft 3,4,5, 1990.

[14]INTEL. *80386/486 System Software Writer's Guide.* Intel, 1987.

[15]INTEL. *i486 Microprocessor Hardware Reference Manual.* Intel, 1990.

[16]INTEL. *i80486 Microprocessor Programer's Reference Manual.* Intel, 1990.

[17]INTEL. *intel486 SX Microprocessor / intel 487 SX Math Coprocessor - Data Book.* Intel, 1991.

[18]KNOBLAUGH, Rick. *Your Own Protected-mode Debugger - A resident debugger for 80386/486/486 platforms.* Dr. Dobb's Journal, September 1992.

[19]LEINECKER, Richard C. *Processor Detection Schemes - You don't have to write for the least-common denominator.* Dr. Dobb's Journal, June 1993.

[20]MARGULIS, Neal. *Advanced 80386/486 Memory Management - Paging is the 80386/486's answer to the memory management challenges for today's multitasking operating systems.* Dr. Dobb's Journal, April 1989.

[21]MORGAN, Christopher L.; WAITE, Mitchell. *8086-8088 Microprocessadores - Hardware.* Makron Books, 1988.

[22]PAPAS, Clovis H. *80386/486 Guia Técnico de Programador.* McGraw Hill, 1989.

[23]PERES, Sandro R. *Sistema de Distribuição de Dados em Tempo Real.* Trabalho de conclusão do Curso de Pós-Graduação em Computação, UFSC, 1989.

[24]SCHULMAN, Andrew; MICHELS, Raymond J.; KYLE, Jim; PATERSON, David; MAXEY, David; BROWN, Ralf. *Undocumented DOS - A Programer's Guide to Reserved MS-DOS Functions and Data Structures.* Addison-Wesley, 1990.

[25]SEGAL, Bernardo; NAKAJUNE, Cesar K.; CELESTINO, Sílvio A. *Conhecendo a Família 80486 - Hardware e Software.* Erica, 1992.

[26]SILBERCHATZ, A. et all. *Operating System Concepts.* Addison-Wesley, 1991.

[27]TANENBAUM, Andews S. *Operating Systems: Design and Implementation.* Prentice-Hall, 1997.

[28]WILLIAMS, Al. *DOS + 386 = 4 Gigabytes! - Directly address 4 gigabytes of memory in DOS from your C or assembly languages applications.* Dr. Dobb's Journal, July 1990.

[29]WILLIAMS, Al. *Roll Your Own DOS Extender: Part I - Develop your own 386 protected-mode applications.* Dr. Dobb's Journal, October,1990.

[30]WILLIAMS, Al. *Roll Your Own DOS Extender: Part II - Under the hood.* Dr. Dobb's Journal, November,1990.

[31]WROBEL, M. *Windows windustrietauglich aufrüsten.* Elektronik, Agosto 1993.

[32]ZAMPIERI, Antonio V. et all. *Solução CEEE para o Sistema de Supervisão e Controle.* CEEE, 1993.

8. Apêndice

8.1 Objetivo

Esse apêndice tem por objetivo apresentar o conteúdo do disquete que acompanha esse trabalho, além de detalhar seu uso.

8.2 O Disquete

O disquete anexo contém:

- O código fonte do protótipo, implementado durante o desenvolvimento deste trabalho, distribuído nos arquivos colocados no diretório FONTES.
- O arquivo executável, resultante da compilação do sistema, colocado no diretório EXECUT.
- A biblioteca de funções de interfaceamento desenvolvidas para a linguagem C, colocada no diretório INTERFAC.
- O arquivo de inclusão contendo as definições e protótipos de funções para linguagem C, também colocado no diretório INTERFAC.
- Exemplos de uso, com código fonte e arquivos executáveis, colocados no diretório EXEMPLOS.
- Uma cópia, em formato texto puro, do material deste apêndice, colocada no diretório raiz, no arquivo LEIA.ME.

8.3 Os Arquivos

O conteúdo dos arquivos que constam do disquete, agrupados de acordo com o diretório onde se encontram, é descrito a seguir.

8.3.1 Diretório FONTES

XDOS386.ASM	Arquivo principal no qual todos os outros são incluídos através do uso da diretiva .INCLUDE
XDOS386.EQU	Parâmetros para o sistema
XDOS386.STR	Estruturas de dados
XDOS386.MAC	Macros
XDOS386.C16	Segmento de código de 16 bits.
XDOS386P.C16	Código de preparação de seletores da GDT, incluído no arquivo XDOS386.C16 através da diretiva .INCLUDE
XDOS386.D16	Segmento de dados de 16 bits

XDOS386.S16	Segmento de pilha de 16 bits
XDOS386.GDT	Segmento com a definição da GDT
XDOS386.ISR	Segmento com rotinas de tratamento de interrupções, Monitor Virtual, Funções de Interface, Escalonador e Rotinas Auxiliares.
XDOS386A.FUN	Arquivo com funções de acesso à interface do sistema, incluído dentro do arquivo XDOS386.ISR através da diretiva .INCLUDE
XDOS386B.FUN	Arquivo com funções de acesso à interface do sistema, incluído dentro do arquivo XDOS386.ISR através da diretiva .INCLUDE
XDOS386B.FUN	Arquivo com funções de acesso à interface do sistema, incluído dentro do arquivo XDOS386.ISR através da diretiva .INCLUDE
XDOS386Z.FUN	Arquivo com rotinas auxiliares para as funções da interface, incluído dentro do arquivo XDOS386.ISR através da diretiva .INCLUDE
XDOS386.ESC	Arquivo com o código do escalonador, incluído dentro do arquivo XDOS386.ISR através da diretiva .INCLUDE
XDOS386.IDT	Segmento de dados com definição dos seletores da IDT
XDOS386.TSS	Declaração dos TSS
XDOS386.C32	Segmento de código de 32 bits
XDOS386.D32	Segmento de dados de 32 bits
XDOS386.S32	Segmento de pilha de 32 bits

8.3.2 Diretório EXECUT

XDOS386.EXE	Arquivo executável com o núcleo multitarefa e código de inicialização e desinstalação

8.3.3 Diretório INTERFAC

XDOS386.H	Arquivo de inclusão contendo as definições globais e os protótipos das funções de interface para linguagem C
XDOS386.LIB	Biblioteca contendo as funções de interface para linguagem C

8.3.4 Diretório EXEMPLOS

PROG1.C	Arquivo fonte do exemplo número 1
PROG2.C	Arquivo fonte do exemplo número 2
PROG1.EXE	Arquivo executável do exemplo número 1
PROG2.EXE	Arquivo executável do exemplo número 2

8.4 Uso do Sistema

A criação de um programa com processos para execução concorrente usando o ambiente XDOS386 passa pelos mesmos passos e utiliza as mesmas ferramentas que a criação de qualquer outro programa. Neste trabalho usamos apenas a linguagem C para criar exemplos e testar o sistema e, portanto, a usaremos para explicar o uso do ambiente.

Os passos a serem seguidos são:

- Análise do problema, com ênfase para processamento concorrente.
- Projeto do sistema, definindo os processos que deverão ser criados.
- Implementação do sistema, usando a linguagem C (em nosso caso).
- Depuração, que se torna mais difícil devido à concorrência.
- Manutenção, tanto corretiva como ampliações nas capacidades inicialmente previstas.

Como o objetivo deste apêndice é mostrar como usa o ambiente, vamos nos ater apenas ao ítem de implementação do sistema, indicando como criar um programa com duas tarefas concorentes.

O esqueleto de um tal programa teria a seguinte aparência:

```
#include <stdio.h>
#include "xdos386.h"

process processo1()
{
.
.
.
}

process processo2()
{
.
.
.
}


main
{
  Instala(processo1,512,"PROC1",1);
  Instala(processo2,512,"PROC2",1);
  Residente();
}
```

Observa-se que os processos são funções normais em linguagem C, declaradas com o tipo 'process', o qual está definido no arquivo de inclusão XDOS386.H. A função 'main'

chama a função de interface 'Instala' que está na biblioteca XDOS386.LIB, para colocar os processos na estrutura de execução concorrente dentro do ambiente. Essa biblioteca deve ser linkada junto com o código objeto de qualquer programa que utilize o ambiente multitarefa.

A função 'Residente' termina o programa principal tornando-o residente na memória DOS. Se não quisermos ocupar a memória do DOS para o programa, podemos usar a função 'Backgrnd', também disponível na interface.

Normalmente, os processos, uma vez instalados, devem ficar executando permanentemente, desde que atendidas as condições de recursos necessários, ou seja, não estando bloqueados em semáforos ou aguardando mensagens. Por isso, o esqueleto de um processo tem o seguinte aspecto.

```
process processo1()
{
// definição de variáveis
.
.
.
// código
  for(;;)
  {
   .
   .
   .
  }
}
```

Nota-se, então, um laço infinito formado por um comando 'for' englobando todo o trecho do processo que deverá ser repetido enquanto o mesmo estiver ativo. No caso de haver necessidade de terminar um processo, um teste deverá ser feito dentro deste laço e a função 'Termina' deverá ser chamada no caso de atendida a condição de término.

8.5 Compiladores Empregados

O sistema descrito por este trabalho foi desenvolvido integralmente em assembler através do uso do Microsoft Macro Assembler, versão 6.1.

Os programas de teste utilizados durante o desenvolvimento e os exemplos que constam do disquete foram desenvolvidos usando a versão 5.1 do compilador C da Microsoft.

8.6 Operação do Sistema

Após executar o programa XDOS386.EXE, o sistema se inicializa e se instala, voltando ao 'prompt' do DOS já executando como processo concorrente (processo SHEL). Todos os comandos do DOS e os programas executáveis são processados normalmente. O sistema exige, para instalação, um processador 80386 ou mais atual, operando no modo real. Caso o processador não seja adequado ou o modo de operação do mesmo não seja o modo real, o sistema emite uma mensagem e termina a execução sem completar a instalação.

Durante a execução de tarefas concorrentes, pode-se verificar que processos estão instalados e sua situação pressinando simultaneamente as teclas ALT e SHIFT esquerdo. Esse procedimento muda a página de vídeo ativa para a página 1 e lista dados dos processos instalados. Para voltar à página normal, página 0, deve-se pressionar simultaneamente as teclas ALT e SHIFT direito.

No caso de ocorrer uma exceção que o sistema não pode tratar, ele se desinstala e volta ao DOS, retirando todos os processos ativos.

8.7 Monitores de Atividade

No arquivo XDOS386.EQU, onde estão as diretivas EQU usadas pelo sistema, existe a definição

.define DEBUGXDOS 1

que controla a exteriorização do que chamamos de Monitores de Atividade, de grande valia quando se está fazendo depuração do ambiente. Se essa linha for transformada em comentário, os Monitores de Atividade não aparecem. Se essa linha for considerada na compilação eles aparecerão.

Esses monitores nada mais são que caracteres que aparecem na tela, em posições definidas e que mudam constantemente a cada chamada de função ou a cada vez que a execução passar pela posição em que o Monitor de Atividade foi colocado. Para acionar o processo, é necessária a recompilação do sistema uando o Assembler da Microsoft, versão 6.1.

No sistema original os Monitores de Atividade estão desativados porém previstos para exteriorização na última linha da tela e posicionados nas seguintes situações:

- As colunas de 0 a 31, sendo usadas para indicar a chamadas de funções da interface, de acordo com a ordem das mesmas na tabela abaixo:

Monitores de Atividade		
Coluna	**Função**	**Descrição**
0	Ativa	Cria estruturas e ativa processo
1	Termina	Libera memória e termina o processo
2	TrocProc	Libera processador para outro processo
3	Identifica	Identifica um processo
4	MonoBegin	Inicia trecho monotarefa
5	MonoEnd	Termina trecho monotarefa
6	InicS	Inicializa uma variavel semáforo
7	P	Executa uma operação P em um semáforo
8	V	Executa uma operação V em um semáforo
9	Send	Envia uma mensagem para um processo
10	Wait	Recebe uma mensagem
11	CriaRel	Cria um relógio para uma rotina
12	ParteRel	Dispara um relógio
13	ParaRel	Pára a contagem de tempo de um relógio
14	CongRel	Congela contagem de tempo de relógio
15	DescRel	Descongela a contagem de um relógio

Monitores de Atividade		
Coluna	**Função**	**Descrição**
16	reservada	Não tem função definida nesta posição
17	reservada	Não tem função definida nesta posição
18	LibRel	Libera relógio para o sistema
19	ProcAtivo	Informa o identificador processo ativo
20	Background	Mantém processos filhos residentes
21	reservada	Não tem função definida nesta posição
22	LibBuf	Libera buffer
23	AlocBuf	Aloca um buffer
24	reservada	Não tem função definida nesta posição
25	Instala	Cria e ativa um processo com prioridade
26	Wake	Coloca mensagem início da fila e acorda
27	Sleep	Espera mensagem de um processo definido
28	Mnemonico	Devolve o mnemônico do processo
29	Residente	Coloca um código residente sem paginar

- As colunas de 33 a 71, tendo sido atribuídas apenas as ímpares, de acordo com a tabela abaixo:

Monitores de Atividade	
Coluna	**Descrição**
33	Interrupção de retorno ao modo protegido após tratar INT8
35	Interrupção de retorno ao modo protegido após executar rotina de relógio de usuário
37	Entrada na função de tratamento da INT21
39	Interior da função de tratamento da INT21, funções 8,7 e 1
41	Interior da função de tratamento da INT21, outra função
43	Acesso ao DOS, antes de ir para função original
45	Acesso ao DOS, depois de retornar da função original
47	Entrada do Monitor Virtual
49	Entrada do Escalonador
51	Chamada de função da Interface do XDOS386
53	Chamada para o sistema ficar residente
55	Inicio da função de desalocação de memória baixa
57	Inicio da função de desalocação de memória alta
59	Inicio da função de alocação de memória baixa
61	Inicio da função de aocação de memória alta
63	? - retorno prot canal
65	Interior da função da INT21, função 31 - TSR
67	Entrada da função de tratamento da INT9
69	Função BloqProc
71	Chamada da função TSR do DOS

8.8 Descrição dos Exemplos

Os exemplos incluídos têm, cada um deles, quatro processos. Cada processo consiste de um 'ventilador' (um quadro com uma barra que gira no interior do mesmo). Todos os

processos tem a mesma prioridade (ou apenas um deles executaria, pois estão sempre prontos) e, portanto pode-se observar que cada um gira durante um tempo e depois pára, enquanto um outro gira. A sequência de execução obedece à ordem de instalação pois todos têm a mesma prioridade e estão em uma fila circular.

Cada um dos programas fica residente através do uso da função 'Backgrnd' que faz uso da paginação, de maneira que, ao carregar cada um deles, não se ocupa a memória do DOS. Faça a seguinte experiência:

- Carregue o ambiente usando o arquivo XDOS386.EXE
- Verifique a quantidade de memória disponível usando o comando MEM do DOS.
- Carregue o primeiro conjunto de processos usando o arquivo PROG1.EXE
- Verifique a quantidade de memória disponível usando o comando MEM do DOS.
- Carregue o segundo conjunto de processos usando o arquivo PROG2.EXE
- Verifique a quantidade de memória disponível usando o comando MEM do DOS.

A cada verificação da memória disponível, o resultado deve ser o mesmo, pois os programas estão sendo executados na área de memória acima de 1 Mb. O tempo de resposta à digitação também aumenta porque o COMMAND.COM, shell do DOS, está sendo executado concorrentemente aos processos instalados.

Para verificar os processos instalados, pressione simultaneamente as teclas ALT e SHIFT esquerdo, e deverá encontrar os oito processos instalados pelos dois programas mais os dois processos do sistema (IDLE e SHEL).

Chama-se a atenção para os conteúdos da coluna onde se encontram os registradores CR3 de cada tarefa. Cada grupo de processos que se encontram em um mesmo programa têm o mesmo CR3. Cada CR3 diferente significa espaços de endereçamento diferente.

Para retornar à tela normal, , pressione simultaneamente as teclas ALT e SHIFT direito.